EPFL

Thèse n° 9608

Uncooperative Rendezvous in Space: Design of an Electronic Architecture for High Performance Avionic with Multi Sensor Input and Intensive Data Rate.

Présentée le 28 septembre 2022

Faculté des sciences et techniques de l'ingénieur
Institut de génie électrique et électronique
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Michaël Yannick JUILLARD

Acceptée sur proposition du jury

Prof. D. Atienza Alonso, président du jury
Prof. J.-P. R. Kneib, directeur de thèse
Prof. J. Eickhoff, rapporteur
Dr Ph. Armbruster, rapporteur
Dr M. Salzmann, rapporteur

École polytechnique fédérale de Lausanne

2022

Curiously enough, the only thing that went through the mind of the bowl of petunias as it fell was *Oh no, not again.* Many people have speculated that if we knew exactly why the bowl of petunias had thought that we would know a lot more about the nature of the Universe than we do now.

— Douglas Adams, *The Hitchhiker's Guide to the Galaxy*

To Loraine and my family...

Acknowledgements

This Ph.D. research would not have been possible without the support and help of several persons.

First, I would like to thank Muriel Richard and Prof. Jean-Paul Kneib for offering me this Ph.D. position at the EPFL Space Center. It was a wonderful opportunity to work on a concrete project and witness the establishment of the ClearSpace start-up. I am grateful as well for their support through these four years and the liberty to pursue the research I wanted.

This research has been funded by Airbus DS GmbH, Friedrichshafen. From Airbus, I wish to thank Prof. Jens Eickhoff and Bharadwaj Chintalapati for welcoming me multiple times to their center. These trips were always very interesting, and I learn so much. An extra thank you to Waj for answering many of my technical emails and doing some remote support from Germany.

I want to thank the members of the committee who agreed to evaluate my work and gave me plenty of relevant feedback, namely: Prof. David Atienza, Prof. Jens Eickhoff, Dr. Mathieu Salzmann, and Dr. Philippe Armbruster.

My thesis was also rhythm with some top-notch assistantship. I am very grateful to Prof. Claude Nicolier, Muriel Richard, and Prof. Bernard Foing for allowing me to support their lecture. These opportunities were highly valuable.

A special thank as well to all my colleagues at eSpace & ClearSpace for the multiple coffee breaks, lunch at the lake, barbecue, ski trips, and so on. My time at EPLF won't have been the same without them. The list is long: Candice, Bastien, Stepan, Simon B., John, Alex, Jacques, Lionel, Tatiana, Hannes, Flavio, Filipo, Elisa, Emmanuelle, Anne-Marlene, Marc-André, Simon H., David, Nicola, Adrian, Mathieu, Denis, Abrahim, Hiro and the many others that spent some time at the center.

Finally, a big thanks to my friends at EPFL and outside for their proofreading and, of course, all the activities we did together. At last, thank you to Loraine and my family for their unconditional support throughout these years.

Lausanne, June 21, 2022 M. J.

Abstract

Active Debris Removal missions consist of sending a satellite in space and removing one or more debris from their current orbit. For these spacecraft, one of the key challenges is to obtain information about the uncooperative target. By gathering the velocity, position, and rotation of the desired object, the satellite is able to plan its trajectory and define the exact sequence of approach. It requires the use of a variety of sensors with often a high data rate. For this task, a dedicated payload computer is envisioned with the responsibility of processing the information from the various rendezvous sensors. This decisive component has the goal to provide meaningful information to the main satellite computer about the targeted debris. The topic of this work is to provide tools and analyses to optimize this payload avionic architecture. The focus is on the data processing, the number of elements, and the electrical energy with constraints due to internal communication.

First, an avionic testbench was built at the EPFL Space Center to assess early hardware and software architectures. The fundamental goal is to enable Hardware-In-the-Loop testing while developing the payload computer. A communication data bus has been implemented between the Platform On-Board Computer and the Payload On-Board Computer (or dedicated payload computer). Emphasis was placed on the reliability of the high-level protocol while establishing a SpaceWire/Ethernet data bus. Additional care was given to the data encapsulation and the information timing. Ultimately, multiple concepts for improvement were analyzed.

In a second phase, the testbench has been extended to support other data buses such as I2C, SPI, CAN, and RS-422. The aim was to develop a reliable and efficient backup or fall-back data bus in case of failure in the main link. These four data buses have been tested with extensive analyses on their resilience to error and the efficiency of their data exchange. In addition, an optimized data compression was elaborated with a focus on minimizing the number of bytes per information. Predominantly, the testing phase indicates the CAN bus is a promising candidate for a reliable fall-back or alternative option.

In parallel with this testbench development, extensive work has been performed to develop a simulation and optimization tool. Its goal is to support the design of payload avionic architectures for ADR missions by providing trade-offs and analyses. In the first iteration, a simulator has been created with instances of various high-level elements modeled. The physical and logical constraints are implemented to dictate the possible actions of the simulated payload

avionic. The output of the simulator displays the behavior of the system over time.

The second iteration introduced the additional dimension of optimization. Not only the tool is able to provide analyses of payload avionic architectures, but it is trying to determine the best set of instruments and algorithms to use. With this capability, extensive analyses have been conducted on the influence of various parameters linked to the general optimizer behavior. It allows us to better understand the strength and limitations of the tool.

The third step regarding the tool was to start its verification. The task has been to develop an Hardware-In-the-Loop architecture to compare its behavior with the results of the optimizer. The implementation of various mock-up algorithms enables the verification of their models in the tool. These analyses guarantee the feasibility of the outputted solution.

The last part was dedicated to the creation and analysis of a realistic payload avionic architecture. The goal was to test the capability of the tool with hardware elements inspired by actual components. This work has shown the procedure to efficiently use the tool in the design phase of a mission.

In conclusion, the work on the communication between the Payload On-Board Computer and the Platform On-Board Computer has brought valuable lessons and experiences to the projects. They can now be used for the establishment of the high-level protocol into ClearSpace-1 flight software. In addition, the optimizer created allows tackling the design of complex payload avionic architecture where mass saving and processing resources are crucial. It is an essential point to develop a highly efficient payload computer for an Active Debris Removal satellite.

Résumé

Une mission pour l'élimination active de débris orbitaux a pour but d'envoyer un satellite dans l'espace et par la suite d'enlever un ou plusieurs déchets. Pour ces engins spatiaux, une des tâches les plus importantes est d'obtenir des informations à propos de leur cible non-coopérative. En obtenant la vitesse, la position et la rotation de l'objet désiré, le satellite est capable de planifier sa trajectoire et de définir précisément sa séquence d'approche. Ces actions nécessitent l'utilisation de nombreux senseurs avec souvent une grande bande passante. Pour cette tâche, un ordinateur dédié à la charge utile a pour responsabilité de traiter toutes les informations venant des capteurs pour le rendez-vous. Le but de ce composant clé est de fournir à l'ordinateur de bord principal des données cruciales à propos du débris ciblé. Le sujet de ce travail de recherche est de fournir des outils et des analyses afin d'optimiser l'architecture électronique de cette charge utile. Cela est fait en considérant les problématiques liées à la puissance de calcul, le nombre d'éléments et l'énergie électrique ainsi que les contraintes communicationnelles entre les cartes électroniques.

Dans un premier temps, un banc de test pour avionique a été créé au centre spatial de l'EPFL pour évaluer des architectures logicielles et matérielles. Son but premier est de permettre des tests avec du matériel incorporé pendant le développement de la charge utile. La tâche a été d'implémenter un bus de communication entre l'ordinateur de bord principal et celui dédié à la charge utile. L'accent a été mis sur la fiabilité de ce protocole haut niveau tout en établissant un bus utilisant les protocoles SpaceWire et Ethernet. Des considérations supplémentaires ont été attribuées à l'encapsulation des données et le minutage du transfert d'information. Pour finir, plusieurs notions pour l'amélioration du protocole ont été analysées.

Dans un second temps, le banc de test a été étendu pour supporter d'autres bus de donnée tel que I2C, SPI, CAN et RS-422. L'objectif était de développer un bus de donnée secondaire fiable et efficace pour soutenir la connexion principale en cas de panne. Ces quatre bus de donnée ont été testés avec des analyses poussées quant à leur résistance aux erreurs et l'efficacité de leur échange de données. Par ailleurs, une méthode de chiffrement optimisée a été élaborée avec comme but de minimiser le nombre d'octets par information. Majoritairement, la phase de test indique que le bus de donnée CAN est un candidat prometteur comme solution alternative ou de soutien.

En parallèle du développement du banc de test, du travail approfondi a été fourni pour dé-

velopper un outil de simulation et d'optimisation. Son but est de supporter la conception d'architectures avioniques dédié à la charge utile pour des missions d'élimination actives de débris orbitaux. Celui-ci a produit des analyses et des compromis sur ces architectures électroniques. Dans une première itération, le simulateur a été créé avec des instances haut niveau de plusieurs éléments. Les parties physiques et logiques des contraintes ont été ajoutées pour imposer les actions possibles de ce simulateur. Ses résultats montrent l'évolution des états du système au cours du temps.

La deuxième itération introduit la dimension d'optimisation. L'outil est non seulement capable de produire des analyses d'architectures avioniques dédiées à la charge utile, mais aussi de déterminer le meilleur ensemble de capteurs et d'algorithmes à utiliser. Avec cette capacité, des analyses étendues sont conduites sur l'influence de divers paramètres liés au comportement général de l'optimiseur. Cela permet de mieux comprendre les forces et limitations de l'outil.

La troisième étape était le début de la verification de l'outil. La tâche consistait à développer une architecture avec du matériel incorporé pour comparer celle-ci avec les résultats de l'optimiseur. L'implémentation de maquettes d'algorithmes a permis la verification de leurs modèles respectifs dans l'outil. Ces analyses garantissent la faisabilité des solutions générées.

La dernière partie était dédiée à al creéation et à l'analyse d'une architecture avionique dédié à la charge La dernière partie était dédiée à la création et à l'analyse d'une architecture avionique dédiée à la charge utile réaliste. Le but était de tester la capacité de l'outil avec des éléments concrets inspirés de vrais produits. Ce travail a introduit la procédure pour une utilisation efficace de l'outil dans la phase de design d'un satellite.

En conclusion, le travail sur la communication entre les cartes électroniques de la charge utile et du satellite a apporté d'importantes leçons au projet. Elles peuvent être maintenant utilisées pour établir un protocole de haut niveau pour le logiciel de bord de ClearSpace-1. Par ailleurs, l'optimiseur créé permet d'aborder les problèmes de conception d'architectures avioniques où des contraintes de poids et de ressources computationnelles existent. C'est un point crucial pour développer de façon efficace un ordinateur de bord dédié à la charge utile d'une mission pour l'élimination active de débris orbitaux.

Contents

Acronyms

ADCS Attitude Determination and Control System.

ADR Active Debris Removal.

AOCS Attitude and Orbit Control System.

ASIC Application-Specific Integrated Circuit.

ASM Assembly language.

ATV Automated Transport Vehicle.

CAN Controller Area Network.

CDPI Combined Data and Power Management Infrastructure.

COTS Commercial Off-The-Shelf.

CPU Central Processing Unit.

CS-1 ClearSpace-1.

DLR German Space Agency.

EDAC Error Detection And Correction.

EPFL École Polytechnique Fédérale de Lausanne.

ESA European Space Agency.

FDIR Failure Detection, Isolation, and Recovery.

FLP 2 Flexible LEO Platform.

FPGA Field-Programmable Gate Array.

FPS Frames Per Second.

FSW Flight Software.

GEO Geosynchronous Equatorial Orbit.

GNC Guidance, Navigation and Control.

GPIO General-Purpose Input/Output.

GPS Global Positioning System.

HIL Hardware-In-the-Loop.

HPC High Priority Command.

I2C Inter-Integrated Circuit.

IAA International Academy of Astronautics.

Acronyms

IP Internet Protocol.

IR Infrared.

ISS International Space Station.

LE Low Energy Consumption.

LEO Low Earth Orbit.

LVDS Low Voltage Differential Signaling.

MINLP Mixed-Integer NonLinear Program.

MIPS Million Instructions Per Second.

MIQCQP Mixed-Integer Quadratically Constrained Quadratic Program.

OBC On-Board Computer.

OOP Object-Oriented Programming.

OS Operating System.

OSI model Open Systems Interconnection model.

PA/QA Product and Quality Assurance.

PCDU Power Control and Distribution Unit.

PF OBC Platform On-Board Computer.

PL OBC Payload On-Board Computer.

PPS Pulse Per Second.

PUS Packet Utilization Standard.

RAM Random Access Memory.

RMAP Remote Memory Access Protocol.

RTEMS Real-Time Executive for Multiprocessor Systems.

RTOS Real-Time Operating System.

SDRAM Synchronous Dynamic Random Access Memory.

SEL Single-Event Latch-Up.

SEU Single-Event Upset.

SPI Serial Peripheral Interface.

SRAM Synchronous Random Access Memory.

TCP Transmission Control Protocol.

TCP/IP Internet protocol suite.

TID Total Ionizing Dose.

TMR Triple Modular Redundancy.

UART Universal Asynchronous Receiver-Transmitter.

UAV Unmanned Aerial Vehicle.

UDP User Datagram Protocol.

XML Extensible Markup Language.

Introduction

The International Academy of Astronautics (IAA) and the European Space Agency (ESA) publish every few years a report on the Space Debris situation[1, 2]. The ESA assessment from 2021 mentions around 28′000 objects (bigger than 10 [cm]) in space with fewer than 7′000 being operational satellites. In comparison, the 2017 IAA report catalogs around 17′700 entities in orbits with merley 1′200 or 6% which are functional spacecraft and about 22% considered as non-debris. By going farther back, in 2006, it was estimated that the earth's orbit contained about 10′000 objects with one-third being non-debris. This massive increase is primarily due to two events. The first one is the destruction of the Chinese *Feng Yun 1C* satellite by an anti-satellite missile in January 2007[3]. This event created 3′433 new debris. The second one in February 2009 is the collision between the *Iridium 33* and *Cosmos 2251* which generated 2′296 debris[4]. Unfortunately, some other major events have happened in the past few years. They can all be identified in Figure 1 from the ESA 2021 report[1].

Not only does current space debris represent an urgent concern; but the situation also keeps deteriorating with the launch of new satellites, therefore increasing the collision probability[5]. Moreover, for every impact happening in orbit, the percentage of debris grows exponentially. Consequently, the probability of collision rises. This chain reaction is equally known as the "Kessler syndrome"[5]. It is estimated that if nothing is done to remove existing space debris and prevent any new ones, this snowball effect could build up a layer of objects around the Earth[5]. It would prevent the launch of new future satellites and by consequence remove the possibility of using any space services like weather, telecommunication, and navigation.

In addition to the number of debris, the size and mass represent a significant point to consider. As it has been monitored in orbit, a satellite or part of it can be shielded against small size objects moving at orbital velocities[6, 7]. Nevertheless, these types of debris are limited to a few grams and one or two centimeters in diameter. Looking at the more massive ones (a few kilograms and up to 30 [cm] in diameter), the realization is that their number is fairly important as well but they are complex to deal with. Their distribution is highly sparse in orbit and they are, of course, difficult to localize. The last class of debris starts from tens of kilograms and is larger than half a meter. Critically, it has been surveyed that removing the five high-risk objects per year would considerably decrease the likelihood of an impactful Kessler

[1] In this Figure, the abbreviations "PF, PD, RB, RF, RD, and RM" refer to some form of space debris. On the other side, PL and PM are active satellites and UI is Unidentified. (Extra information is provided in the ESA report[2].)

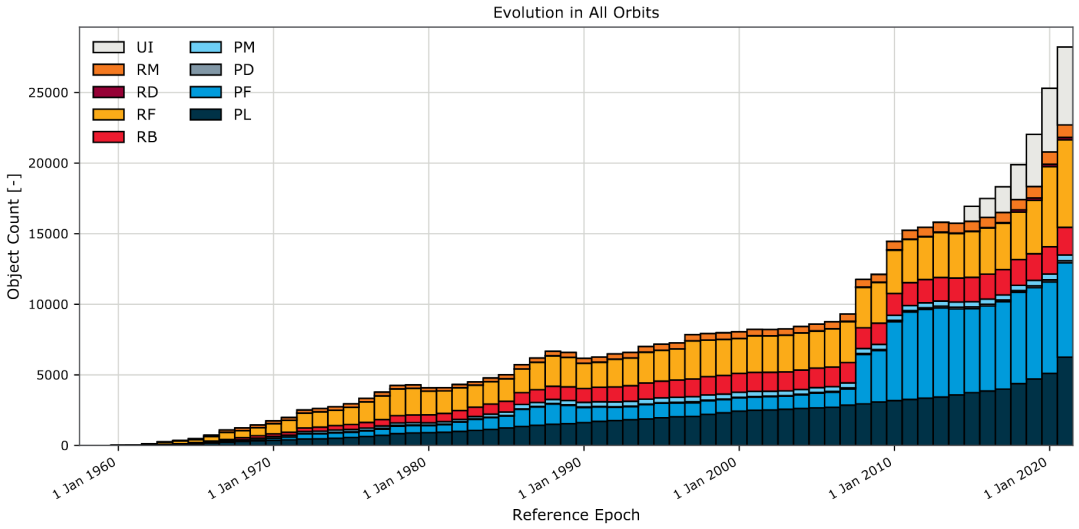


Figure 1 – **Space objects over time[2] ©ESA Space Debris Office.**

syndrome[8]. Figure 2 shows the mass reparation in Low Earth Orbit (LEO) in 2020. More than 40% are not active satellites and can lead to potentially catastrophic collisions.

As an answer to this growing challenge, various entities have started working on the topic. One of the key aspects is Active Debris Removal (ADR) space missions. Because debris' size can range from centimeters to meters, it is complex to conceive a one-size-fits-all solution and thus design choices have to be made to pick a category. The challenge is not the same to aim for a 1 $[tonne]$ object with a large structure or an exploded satellite remaining with screws and pieces of metals. As shown before, the lighter ones are less fundamental and thus this category is typically not targeted. Regarding the medium ones, it is nowadays unfeasible to launch a dedicated satellite for uniquely a few of these debris. The mission architecture has to focus on deorbiting a large amount of them at the same time. Some concepts have been already presented[9] but demonstrators need to be launched. Because of their criticality and their potentially devastating impact on the in-orbit population, the deorbiting of a large object is an essential capability to develop. This implication leads to the creation of the ADRIOS mission with its ClearSpace-1 (CS-1) satellite. Its goal is to deorbit a relatively large object in the coming years.

ADR missions can be considered as a mix between formation flying and docking but with additional constraints. The lack of information regarding the target and the uncooperativeness of the object are two of the primary ones. The first issue implies there is little knowledge regarding the state and rotation of the debris. Although optical ground tracking provides estimations of the attitude (orientation of an object in a specific reference frame) and the shape of large satellites, it is impossible to resolve for small objects < 2 $[m]$. What can be detected from the ground is the light variation that is reflected by the sun and its frequency. Nevertheless, because of possible degeneracies, primarily in-orbit pose estimation can produce a precise

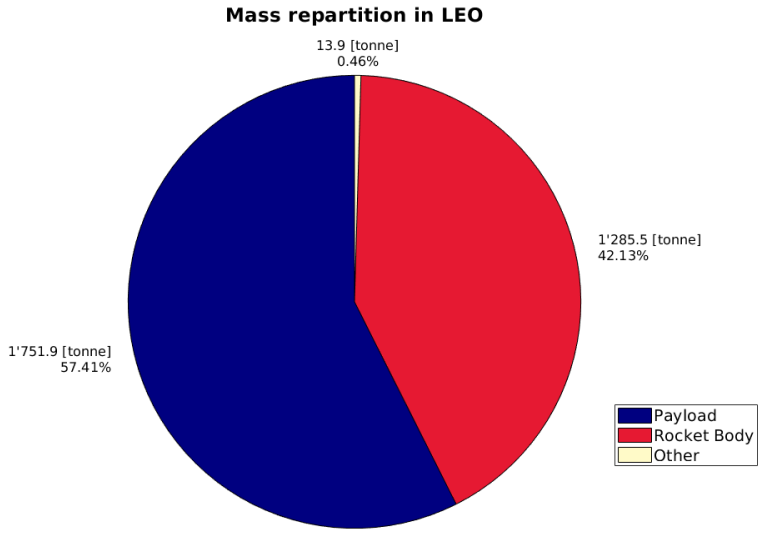


Figure 2 – **Mass repartition in LEO (2020). Numbers from ESA report[2].**

answer. If the light variation is low (e.g. when the rotation is slow) the degeneracies tend to be larger. In summary, it implies dedicated sensors embedded into the satellite and a high amount of processing resources available to compute the various information. For these reasons, the avionic of an ADR mission is crucial to gather key data about the object. In addition, the spacecraft needs to be reactive and has to feature a high level of autonomy, especially in the final approach phase.

A Motivation

Active Debris Removal (ADR) in space remains nonetheless a topic in its early development phase. Some demonstrator missions have been launched or are being developed, but no real debris in LEO has been removed at the time of writing this thesis. The initial task of the satellite (chaser) is to detect and track the targeted debris or object, then perform some proximity operations before capturing it. Associated with each phase, there are a few unique challenges mainly linked to the uncooperativeness of the target. To handle these problems, the chaser must embark a variety of sensors for the Guidance, Navigation and Control (GNC). It must equally have multiple dedicated algorithms for the pose and attitude estimation of the target. To obtain accurate information, the algorithms require a high input data rate and from multiple sensor sources. The first one guarantees constant tracking and the second prevents any biasing. Furthermore, because of orbital mechanisms and potential low ground coverage, data have to be analyzed onboard to satisfy a constant feed of the algorithms.

The avionic (or onboard electronic) of such missions remains by default a crucial point. To start, the chaser needs an array of sensors to be able to observe and detect the target at various distances. From tenths of kilometers away, the spacecraft would include some sensitive instruments such as cameras to be able to detect and track the targeted debris. In contrast at around 5 $[m]$ behind the target, the sensors would require some wide-angle and high definition to inspect the object. Additionally, the issue of motion reconstructions is significant in the case of a tumbling target and would involve stereo imaging and a potential Lidar. For some phases, the illumination can produce saturation in the sensors and prevent correct tracking. Besides, the "night" time phase would be challenging for standard cameras and might require IR sensors or RADAR. In an ideal world, the spacecraft would carry each type of instrument and some redundancy to ensure the success of the mission. However, it considerably increases the complexity of the missions as well as the mass and cost of the chaser. Compromises have to be made.

The second challenge comes in the handling of the data. The task needed to be done can be compared to some space telescopes or earth observation satellites. Indeed, most of the data processing has to be performed in orbit to prevent the bottleneck of the transmission. It would be a challenge to download all the information acquired by the sensors down to Earth without a large antenna and a powerful transceiver. To reduce this part, onboard data processing is crucial. In addition, the connection to the ground stations might be delicate or impossible in some phases and therefore make the transmission problem even bigger. Nonetheless, transmitting some raw data down to Earth remains a key point to verify the integrity of the various algorithm. Naturally, the decrease in communication implies a more substantial difficulty for the avionic. In addition to this processing, another issue lies in the complexity of the various algorithm. Their task is to compute meaningful information from measurements. For example, an image processing action would extract the pose of the debris every second and a second algorithm would compute its general motion. Evidently, these processes could be regrouped under one designated algorithm[2]. Their ultimate purpose is to feed the navigation system of the spacecraft to achieve the rendezvous and capture phase of the debris. The underlying challenge is to achieve a compromise between the amount of information computed and the complexity of the algorithms.

The last essential objective remains the ability of the Payload On-Board Computer (PL OBC) to handle all the processing and computation. As seen formerly, each sensor might produce a large amount of data. It implies the processor has to handle and redirect it to the correct algorithm. The idea of operating a dedicated computer (PL OBC) is decisive. This component decouples the primary avionic function of a satellite from the specific task of the mission. Additionally, this extra computer allows employing optimized electronics hardware to manage the processing and reduces the risk on the main avionic. Nevertheless, it implies handling a second complex system and having two computers talking to each other constantly.

Figure 3 shows a generic view of an ADR satellite with emphasis on important elements

[2] It will be assumed therefore in this thesis.

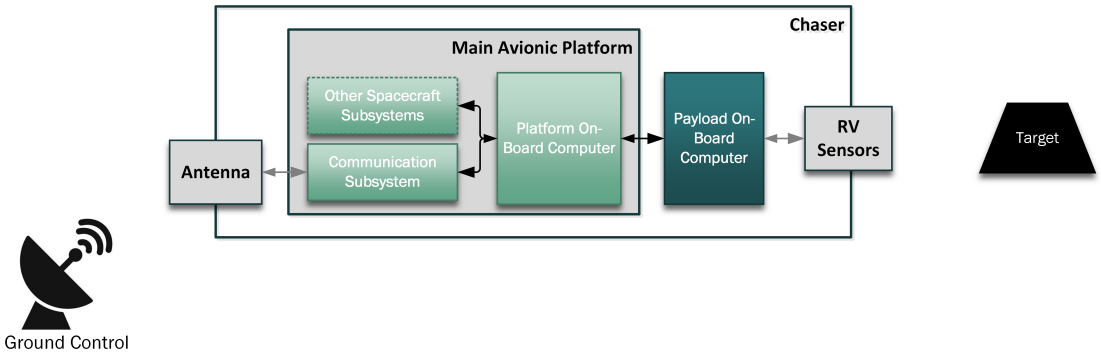


Figure 3 – **Simplified view of an Active Debris Removal satellite with focus on the PL OBC.**

addressed during this thesis. As it can be seen, fundamental elements such as the power subsystem or the GNC subsystem are not shown. The key point is to highlight the Payload On-Board Computer and its connection to the rest of the spacecraft.

In summary, the avionic of an ADR mission faces three fundamental constraints. First, the number and types of sensors to gather the proper measurements at the appropriate moment. Second, the complexity and type of algorithm to compute enough key information required by the navigation system. Third, a powerful and flexible payload computer (PL OBC) to accommodate the previous elements and communicate with the Platform On-Board Computer (PF OBC). In this thesis, these challenges are tackled through early software testing in a Hardware-In-the-Loop (HIL) setup and the creation of an optimization tool for initial payload avionic design.

B State Of the Art

B.1 Relevant Space Mission and Project

It is capital to acknowledge the challenges of the various past and ongoing space missions to develop a proper avionic that suits the need of an ADR satellite such as CS-1. The most interesting categories of missions are, of course, existing Active Debris Removal like RemoveDEBRIS [10], e.Deorbit [11], NanoRacks-Remove Debris [12], and ELSA-d [13]. Similar types of problems can equally be encountered with formation flying spacecraft considering an almost autonomous approach like the ATV[14, 15], PRISMA [16], or the AVANTI experiment [17].

Among other relevant mission types, there are space telescopes with their high processing rate and commercial avionic platform with their necessary reliability. Finally, multiple research works have been done on exotic architecture and Field-Programmable Gate Array (FPGA) in space to improve the performances of avionics.

In addition, the autonomous industry has worked on similar problems for many years. The rise of autonomous cars, ships, and drones has already imposed certain standards in the industry.

Moreover, the need for highly reliable vehicles in an often non-predictable situation has set the requirements for the performances. The drone industry remains an excellent example regarding miniaturization and optimization of the hardware.

The goal of the review is to present various missions, projects, and research that needed specific hardware architecture. As mentioned, emphasis is made on space missions and projects with high processing needs and/or complex navigation algorithms to better acknowledge the challenges of the ClearSpace-1 Payload On-Board Computer (PL OBC).

Formation Flying & Active Debris Removal

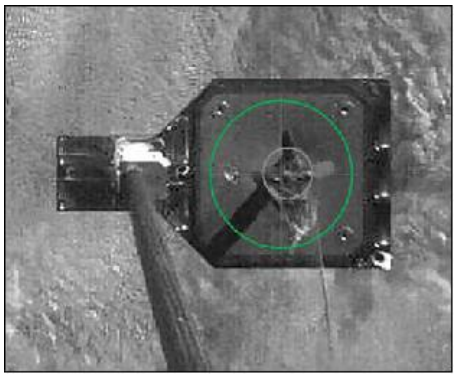
The Automated Transport Vehicle (ATV) [14], [15] project led by the European Space Agency (ESA) was created in the late nineties to resupply the International Space Station (ISS). Ultimately, five vehicles have flown, and each of them was based on a similar design, but every iteration brought few improvements. Its essential characteristic remained its ability to almost automatically dock the ISS (Figure 4a) by using a set of different sensors.

The ATV can be considered an excellent starting point to comprehend the implication of designing complex avionics. Indeed, it needs to handle both a high level of safety and precise navigation. One of the most emphasized points in these publications is the safety of the vehicle and its ability to abort the final approach with up to 2 failures[18]. To be more precise, the ATV could maintain its nominal operation in case of a single hardware failure. If a second failure happens, the vehicle could still operate an abort maneuver autonomously[19].

Another challenge of the missions was the rendezvous sensors used during the approach. From free-flying to 250 [m] behind the ISS, it was performed with the relative Global Positioning System (GPS) information from the ISS and the vehicle itself. For the final, they exclusively employed the Videometers instruments which were providing range, line-of-sight, and relative altitude[20]. It was based on Star Tracker technologies and used Laser Retro Reflectors in diverse patterns mounted on the docking port of the ISS. The instrument composed of laser diodes and rotating mirrors was providing range, range rate, and line-of-sight from 300 [m] for the relative navigation filter. To increase safety, two Telegiometers were used solely to monitor the final phase and were decoupled from the navigation algorithm[21]. A 3D imaging Lidar was also implemented (but not used) during the last mission to further increase the capabilities of the spacecraft[22]. Most of the control loops operated at 1 [Hz] but some of the instruments provided data at 10 [Hz].

Regarding the flight control design, the GNC Measurement System Functional Unit gathered the information of the different sensors[23]. In addition, there was the Frame reference Software Unit[24] that computed the local orbital reference frame orientation and the Sun direction. These units subsequently gave their information to the Guidance, Navigation and Control Software Unit. It computed the ATV kinematics state and the aimed kinematic orders. The propulsion Functional Unit was then responsible to provide the requested force and

(a) ATV-5 docking[27] ©NASA,ESA



(b) RemoveDEBRIS harpoon demonstration[28] ©RemoveDebris Team

Figure 4 – **ATV-5 and RemoveDEBRIS missions.**

torques to the propulsion system[14]. Concerning the hardware architecture, the ATV avionics was fully redundant in all part[25]. The avionic utilized four main fault-tolerant computers and each one was connected to four system buses[26]. Each computer was designed to monitor one bus. Ultimately, virtually every module included two instances that were redundant, and both of them were connected to two unique buses.

Another fundamental mission is the PRISMA[16] (Prototype Research Instruments and Space Mission Technology Advancement) project by the Swedish Space Corporation. It started in 2005 and launched in 2010. The goal was to demonstrate formation flying and rendezvous technologies for small satellites. The project included the participation of the German Space Agency (DLR), the French Space Agency (CNES), the Technical University of Denmark (DTU), and other companies. The mission included two small satellites, respectively identified as Mango and Tango. It conclusively demonstrated various technologies like relative GPS navigation, radio frequency sensors for ranging, and vision-based sensors with a Star Tracker.

The decisive point of the mission was the ability to perform an autonomous rendezvous with a satellite the size of CS-1. Moreover, it was using sensors that will probably be similar to the ones employed by an ADR mission. The Mango satellite (chaser) was the one doing the maneuvering and the rendezvous. It was a 150 $[kg]$ spacecraft with two redundant onboard computers and a variety of sensors. Its main computer boards were based on a LEON processor and handled all the software including the GNC[29, 30]. The consortium wanted to achieve several objectives such as formation flying, autonomous rendezvous and proximity operation, and all of them needed to use the processors aforementioned. Each of the phases included its own set of algorithms described in a specific chapter of the book "Distributed Space Missions for Earth System Monitoring"[31]. This book provides also supplementary information about the hardware used. The data handling system is based on a LEON3 microprocessor with a 32-bit SPARC V8 architecture. It is a fault-tolerant processor and is implemented on a FPGA which provides 20 Million Instructions Per Second (MIPS). Additionally, it contains a floating-point

unit and operates a CAN bus to communicate with the rest of the spacecraft.

The following crucial step in the domain is the AVANTI[17] (Autonomous Vision Approach Navigation and Target Identification) experiment by the DLR. The project was launched in 2016 inside the BIROS satellite, part of the FireBird mission. The goal of this secondary scientific experiment was to demonstrate relative orbit determination with Angle-Only (AO) navigation, maneuver planning, and onboard safety monitoring. It used the CubeSat BEESAT-4 which was carried by BIROS for its uncooperative rendezvous experiment. Its star trackers were used for vision-based navigation from far- to mid-range to demonstrate autonomous rendezvous. This experiment confirms the feasibility of relative navigation with a unique sensor and low data rate. Even though this approach cannot be adopted by the CS-1 mission, the lessons and the procedure are important.

As explained earlier, the AVANTI experiment[17, 32, 33] was part of the BIROS satellite. The spacecraft itself was based on an avionic platform called TET-X (direct upgrade of the TET-1) and built by OHB Sweden[34, 35]. The satellite's architecture was composed of four redundant computers that managed every aspect of the spacecraft. Furthermore, there were watchdog circuits for the Failure Detection, Isolation, and Recovery (FDIR) part. Significantly, BIROS did not have a Floating Point Unit. Two separated nodes, respectively "worker" and "master" nodes, were created inside the computer. The first one controlled the satellite and the second one supervised the proper operation. The AVANTI experiment could also manage the Pre-Processing Unit (PPU) memory (80 [MB]) designed for the other BIROS experiment. As previously mentioned, the AVANTI experiment used "standard" sensors for its relative navigation. Concerning the GNC, the algorithms were based on an analytic model and were updated every 30 [s].

Another mission in the domain is RemoveDEBRIS [10] which included Airbus, ArianeGroup, CSEM, SSTL, and other actors. It was launched in April 2018 to the ISS and later released from it. The mission includes a net experiment, vision-based navigation, target inspection with a Lidar, multiple optical cameras, a harpoon (Figure 4b), and also a dragsail. Most of these technologies would be beneficial for the CS-1 satellite, especially the vision-based navigation and the target inspection. The very fascinating results of the mission and the operation can be found in one of their last publication[36].

The most known ADR mission is likely e.Deorbit [11] lead by European Space Agency (ESA) which was planned to be launched in 2025. Even though the mission funding has been stopped, many highly interesting developments have been done. Part of this work was the HIPNOS (HIgh Performance avionics solutioN for advanced and complex GNC Systems) project [37] which aimed to develop avionics and algorithms for ADR missions. The project's goal was to move out of the space-grade processor (LEON3 or RAD750) and use non-Rad-hard Commercial Off-The-Shelf (COTS) components to lower development prices and increase performance. Moreover, because of the short time in orbit, this alternative was attractive. Their vision was to employ a System-On-Chip device with processors and FPGAs integrated.

The hardware used for their development was the Zynq MMp board with an FPGA and a dual-Core ARM Cortex Central Processing Unit (CPU) [37].

In an identical idea, the GMV company[38] has carry out a similar analysis but for near-earth object (NEO) asteroid rendezvous[39, 40]. In their research, they reached a similar conclusion stating a combination of FPGAs and processors will give the highest performances and scalability. They aim to have a Real-Time Executive for Multiprocessor Systems (RTEMS) real-time operating system with GNC algorithm coded in C implemented in a LEON2 processor. For the image processing algorithm, a direct hardware implementation inside the FPGA is required. Concerning the rate, they estimate both the image processing and the navigation filter to run at 1 $[Hz]$.

Regarding newer demonstration missions, it is critical to mention the ELSA-d satellite from Astroscale [13]. It was launched in March 2021 and is currently in its operation phase. The goal is to demonstrate rendezvous operation and capture mechanism with a small target. Their demonstration mission is extremely interesting because of their maneuver for close rendezvous and the performance of their navigation system. The essential difference remains the use of a magnetic docking plate for the capture mechanism limiting the type of objects they can catch and ruling out most space debris. They have successfully tested their system in August 2021[41].

As a surprise to many, China has demonstrated the removal of one of their Geosynchronous Equatorial Orbit (GEO) satellite in January 2022[42]. The Shijian-21 has been launched probably in mid-2021 and has been inspecting its target (Beidou-2 G2 navigation satellite) since December of the same year. The ADR satellite has docked or capture its target and towed it $300'$ $[km]$ above the Geosynchronous Equatorial Orbit (GEO). The impressive part is that this satellite is now back in GEO. Unfortunately, no precise information was found regarding the ADR spacecraft.

Among the Active Debris Removal missions, many other concepts have been presented in the past. There is the Front-End Robotics Enabling Near-Term Demonstration (FREND) Robotic Arm[12] which aims to be a 7 DOF system to catch and handle spacecraft and potential debris. The idea to use laser or ion propulsion to deorbit smaller debris has been around for a while. M. Schmitz et al.[43] suggest the use of a laser to ablate part of the debris which creates thrust. They aim at objects in the range of $1-10$ $[m]$. A similar concept is presented by C. Bombardelli and J. Peláez[44] in which they suggest the use of an ion beam to apply a force on the other satellite. Another idea is to use expanding foam injected around the debris to increase their area-to-mass ratio and thus the atmospheric drag[45]. More information on the current trend of ADR missions concepts can be found in "Review of active space debris removal methods" (2019)[9] and "Active debris removal: Recent progress and current trends" (2013)[46].

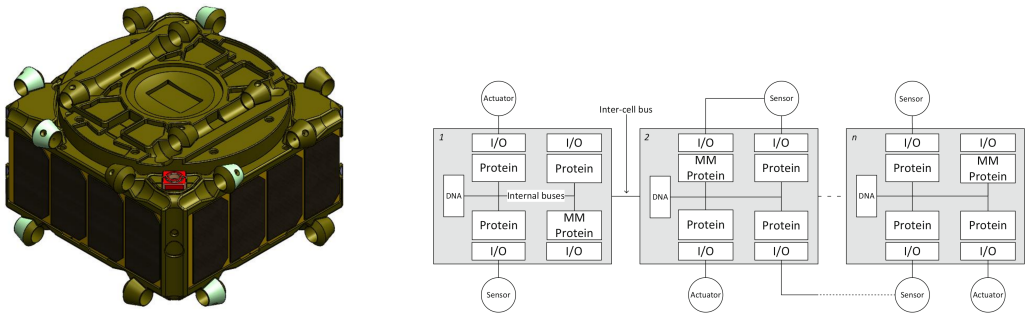
Company	Platform	Processor	Comment
IRS, University of Stuttgart	FLP	2x RadHard LEON3 CPU UT699	Hot redudant RTEMS RTOS[48]
Airbus DS GmbH	FLP 2	2x dual-core LEON3 GR712[49]	RTEMS RTOS[48]
Airbus DS GmbH	Eurostar serie	Mil-Std-1750	[50]
Surrey Satellite Technology LLC	SST-X00 series	PowerPC 750FL	Designed by IBM RTEMS RTOS[51]
SSL (Space Systems/Loral)	SSL 100 & SSL 3000	Rad750	[52]
OHB	SmartLEO, SmartMEO, etc.	LEON2	Fault-tolerant[53]
Lockheed Martin	LM series	RAD750	Fault-tolerant[54]
Boeing (BDS)	BSS-702X BSS-601X	Spacecraft Control Processor	[55]
Thales Alenia Space + EADS	Alphabus	LEON3FT	[56, 57]
Northrop Grumman	GEOStar serie	BAE RAD750	[58]

Table 1 – **Platform summary**

Commercial Avionic Platform and Space Telescope

In the space mission, there are different "platforms" developed by industry. These platforms are standardized avionic which are provided to universities and other companies to facilitate the creation of satellites. In this category, there is obviously the Flexible LEO Platform (FLP 2) by Airbus DS GmbH, Friedrichshafen[47]. Another platform based on the OneWeb constellation satellite is being produced by Airbus & OneWeb. There are also many other manufacturers like SSTL, OHB, SSL, Boeing, Thales, Lockhead Martin, Orbital ATK (now part of Northrop Grumman). Each of them has a specific power interface and navigation system, but the concept and the general architecture remain similar. Most of them are often designed for earth-orbit with differentiation between LEO and GEO. It should also be stated that these platforms are primarily designed for telecommunication and earth observations satellites and might not accommodate the CS-1 requirements. The table 1 is a non-exhaustive summary of the various platforms and their processor. As it can be observed, the main processor tends to be either from the LEON or the RAD750 family.

The last type of relevant mission to talk about concerns space telescopes. For simplicity, two of them are studied. Because of their high-quality image and high data rate, they have allowed improvements in the onboard image processing algorithms and data compression. In this category, there are two European satellites: GAIA[59] launched in 2013 and its successor EUCLID[60] who will be launched in 2023. Both process the spatial resolution on-board before transmitting back to earth where temporal resolution is done.



(a) Satlet unit (DARPA – US DoD)[63]

(b) StemCell concept (Surrey University)[64]

Figure 5 – **Two exotic avionic concepts.**

Concerning the architecture of the GAIA space telescope, the command and data handling uses an ERC-32 processor (radiation-tolerant 32-bit RISC) for the central computer. It communicates with the Payload Data Handling Unit (PDHU) which itself has seven redundant SpaceWire (40 $[Mbit/s]$ each) connected to the Video processing Units (VPU). Each VPU has a processing capability of 1 $[GIPS]$ and a compression factor of 2.8. The PDHU also hosts a one Terabit mass memory[59, 61].

Concerning Euclid, the architecture is a bit unique. Regarding the main avionic, it has two redundant LEON Fault-tolerant processors that have a computational power of 40 $[MIPS]$ and 5 $[MFLOPS]$. In addition, it has a 4 $[Tbit]$ mass memory directly connected to the two instruments (visible imager VIS and Near-Infrared Spectro Photometer NISP) with SpaceWire. The VIS has a 144 CCD detector and produces an image of $24k \times 24k$ pixels. This image is compressed in 250 $[s]$ by a Maxwell SCS750 PowerPC triple-voting processing unit. The NISP has a LEON2-FT CPU embedded in an Application-Specific Integrated Circuit (ASIC) with also a FPGA to improve its performances[60, 62].

Exotic Avionic Architecture and FPGA in Space

In the exotic type of space avionic architecture, there is the "Phoenix Satlet" developed by NovaWurks, Inc.[3] and funded by the Defense Advanced Research Project Agency-DARPA (part of US DoD). The idea behind these Satlets is to produce half-CubeSat units that contain all the functionality of a satellite like the main computer, Attitude and Orbit Control System (AOCS), thermal and solar panels. A model can be seen in Figure 5a. The fascinating part remains the ability to stack several together (up to 30) to achieve greater performance. In the HISat experiment, they want to have 14 Satlets connected. Each one has a microprocessor with a $2-10$ $[GIPS]$ capability, 1 $[GB]$ of Random Access Memory (RAM), and 32 $[GB]$ of flash memory. It results in a highly distributed system where each processor needs to share its resources. They also operate tests where some of these Satlets "fail" to ensure the guarantee of the nominal operation[63, 65, 66].

[3] NovaWurks, Inc., 10772 Noel Street, Los Alamitos, CA 90720-2548

Another approach to distributive architecture is the concept developed by Surrey Space Center and named "Stem Cell" or "Multicellular Architecture". The idea is to increase the performance and flexibility of the architecture by creating artificial cells implemented with FPGAs and microprocessors. The system is composed of numerous cells which have multiple microprocessors and accelerators. Task management is shared between all the cells and various processes. They can assign and/or modify tasks while the whole architecture is running. The concept is shown in Figure 5b. Moreover, they claim to be resilient against cell failure by redistributing tasks in a short reaction time. In the latest publication, they proposed an implementation of their design for a CubeSat. One cell is composed of four ARM Cortex microcontrollers running at 48 [MHz] giving an overall 120.96 [$DMIPS$]. They communicate using CAN and I2C inside cells and CAN between cells. In addition, each element has a 256 [kB] EEPROM memory[64, 67, 68, 69, 70].

The following paragraphs are dedicated to the research carried out on FPGAs and space application. In this field, extensive studies have been done to ensure the reliability of reprogrammable FPGAs. Indeed, if a transient fault happens, it alternates the logic implemented, therefore making the whole hardware faulty. Transient faults typically happen in space due to high energetic particles going through the electronics. The fundamental issue of these faults is their accumulation over time, making them almost impossible to detect and correct without specific methods. Few techniques have been discovered to ensure the fault tolerance of the system and the most known one is the Triple Modular Redundancy (TMR)[71]. The principle is to implement three times the same logic in parallel and utilize a voting method at the output to select the most "probable" answer. This technique can be exploited by the logic itself up to an entire system, however, the downside remains the increase of area used by a factor of 3. Variation and improvement of these techniques can be found in the literature with examples like the combination of hardware and time redundancy[72]. In addition, there is the reprogramming FPGA constantly to prevent error accumulation[73] and special routing algorithms to avoid multiple errors due to one transient fault[74]. Ultimately, the list includes different techniques than FPGA to increase the tolerance to fault [75] and trade-off to find at which level to implement FPGA [76]. In terms of avionic architecture, there are also various publications with FPGAs used directly with an On-Board Computer (OBC) included in FPGAs[77, 78, 79, 80].

F. Lima et al.[72] proposed a variant of this technique by combining the hardware redundancy with a time redundancy. By merging both of them with different voters, they are capable to restrict the size increase by a factor of 2 and with a slight reduction in timing performances. In another publication[73] the same authors describe few design points regarding the implementation of such FPGA techniques. They emphasize the majority voter fault tolerance as well as the need to reprogram at a certain rate the FPGA to prevent error accumulation (Scrubbing). Pratt et al.[81] present an alternative to fully TMR FPGA by exploring the sensitivity of bit in few designs. They propose a trade-off by protecting with FPGA uniquely the most sensitive part of a design. Sterpone & Violante[74] explore the issue of the FPGA routing. They discover that certain faults were creating multiple errors in the design. As a solution, the authors suggest a

new algorithm to minimize the impact of such an effect.

Lovelly et al.[82] published a study in 2014 presenting the state of space processors and forthcoming trends. This publication provides a broad range of quantitative comparisons of various processors. Designs and ideas of more classical avionic architectures targeting real-time and high performance can be found in many publications[83, 84, 85, 86, 87]. The low-cost aspect of performant space electronics is additionally something widely looked at, especially with the usage of COTS components[88, 89, 90, 91].

Terrestrial Drone

On the ground, many technologies concerning autonomous navigation have been developed in the past years. One of the leading categories is the drone industry with its miniaturization, data fusion, and precise navigation. Among the projects, both the research and the industry can be explored. Few cases can be examined to better understand the context. In the research, many universities are working on this topic and especially ETHZ with the *Autonomous Systems Lab* and the *Vision for Robotics Lab* and EPFL with the *Laboratory of Intelligent Systems*. All these laboratories possess an interest in the vision-based perception for drones and also robots. They are repeatedly working on small drones which imply limited power and mass. In addition, they have the challenge of implementing different sensors and algorithms for advanced vision with everything running in real-time[92, 93]. On the industry side, companies like DJI[94] or Parrot[95] can be cited. Concerning the market, there are not only leisure drones but the ones for autonomous inspections and other services. For both cases, the goal is consistently to have smooth control and the best performances while achieving greater autonomy. These developments from research institutes and industries are beneficial for space application, especially in the higher performance domains and miniaturization. However, it needs to be remembered that these technologies are not space qualified and the orbital dynamics apply different constraints to vehicles.

The PIXHAWK drone[93, 96] designed by the Computer Vision and Geometry Lab at ETHZ is a drone that has an onboard vision computer that works in collaboration with its navigation control. The computer vision algorithm is implemented in an Intel Core 2 i7 running at 1.88 $[GHz]$. It has two main image processing algorithms operating at 15 $[Hz]$ and 30 $[Hz]$. The main drone controller used to run on a 60 $[MHz]$ ARM7 microcontroller with the navigation algorithm and now has been upgraded to an STM32F4. The image processing board provides information for the navigation to improve performance.

The University of Tübingen in Germany has developed an external processing board for vision algorithms that is directly connected to a standardized drone[92]. The idea is to improve the performance of commercial drones by including specific hardware. They use an AR.Drone that has a main processing unit with an ARM9 processor running at 468 $[MHz]$ (or ARM Cortex-A8 at 1 $[GHz]$ for the second version). This board is connected with Universal Asynchronous Receiver-Transmitter (UART) to an external processing unit. They have investigated three

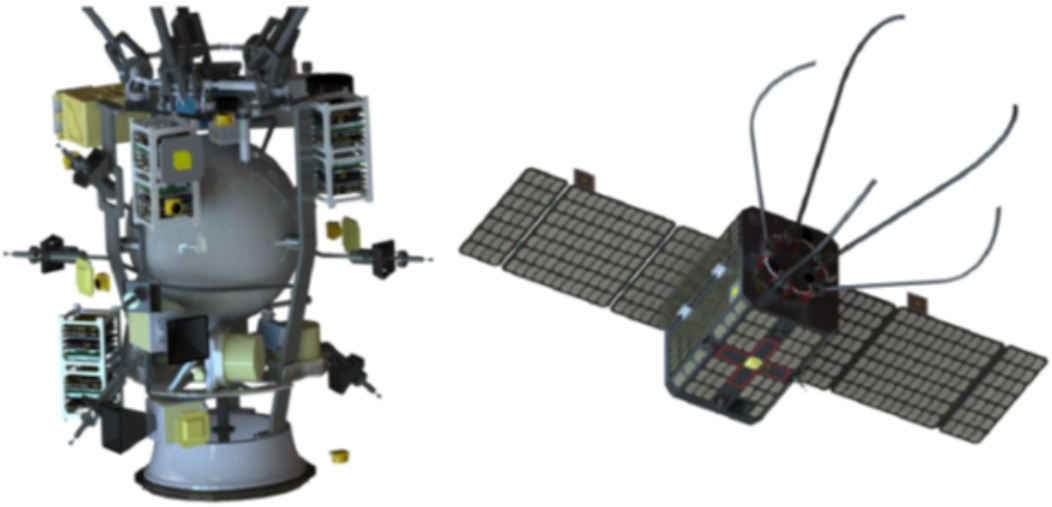


Figure 6 – **Preliminary concept of CSO[99].**

various boards: 14.74 [MHz] ATMEL Atmega1284 8-bits microcontroller, 700 [MHz] ARM Cortex-A8 processor or 1.7 [GHz] ARM Cortex-A9 quad-core processor. The autonomous navigation, control algorithm, and computer vision part are implemented in the additional hardware. The main processor board handles uniquely the pose estimation.

The Queensland University of Technology in Australia also offers a similar approach to improving drone performances[97]. Their approach is to employ an embedded FPGA to implement a path planner in a "commercial" drone. In their study, they use an AR.Drone Quadcopter from Parrot using a PX4 autopilot board. This board is composed of a 32-bit ARM9 RISC processor and is then linked to their custom FPGA with an UART connection. The FPGA is a Xilinx Zynq 7-Z010 unit that has 28k logic cells and 17.6k LUTs. They are developing their platform for many ground applications such as Search and Rescue[98].

In conclusion, the tendency on the research side is to combine existing hardware and controllers with custom parts. The goal of these extra hardware parts in all cases is to improve the general performances of the drones. Similarities can be drawn between their approach and the one targeted by satellite manufacturers.

B.2 From ClearSpace-1 Mission to the Hardware-In-the-Loop Testbench

ClearSpace-1

ClearSpace-1 (CS-1)[100](Figure 7) is a mission developed by the start-up company ClearSpace together with the EPFL Space Center. The goal is to rendezvous with VESPUP (VEga Secondary Payload Adapter Upper Part) and deorbit it. A similar mission was being designed by the start-up in an initial phase, and it was designated as CleanSpace One (CSO) [99] (Figure 6).

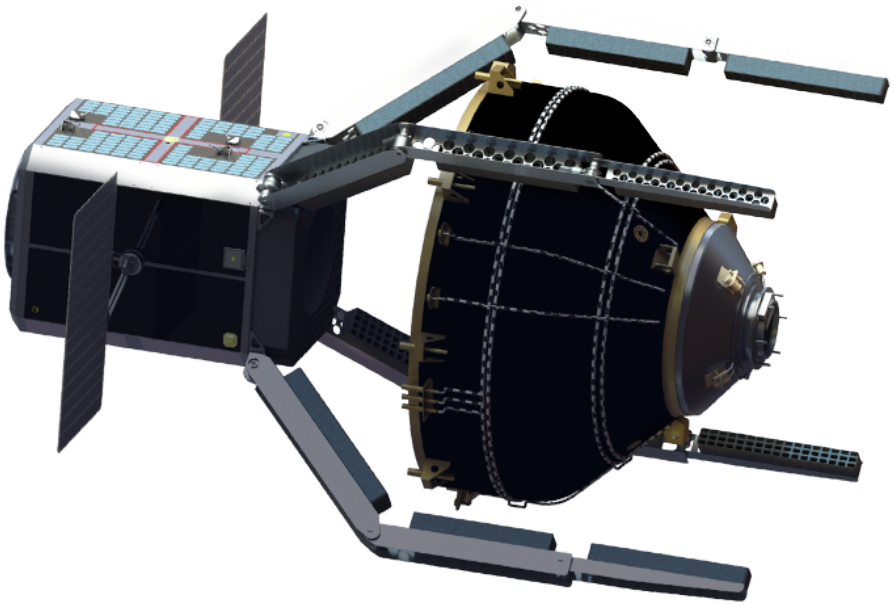


Figure 7 – **Some concept of CS-1 [100] ©ClearSpace Team.**

CS-1 will demonstrate the technologies needed for future ADR missions. The development focuses on three primary aspects. The first one is a capture mechanism able to retract and deploy multiple times, moreover, it should perform a soft capture of the target. The second aspect is about the various algorithms needed by the satellite to find, track and estimate the 6D pose of the debris. The third crucial development is the creation of a Payload On-Board Computer (PL OBC or simply payload computer) to host the previously mentioned algorithm such as relative navigation and image processing algorithms. It should merge the data from various sensors needed for the approach and rendezvous phases.

To achieve a high degree of confidence during the final approach phase, the spacecraft needs to hold a variety of sensors with a diverse spectrum of range and accuracy. They feed the algorithms running in the satellite's PL OBC which compute accurate information about the target. To guarantee constant tracking and prevent any biasing of the pose estimation, the algorithms require high input data rates from multiple sensor sources. Furthermore, because of orbital mechanisms and potentially low ground coverage, data have to be analyzed on-board to satisfy a constant control of navigation. For a similar reason, Collision Avoidance Maneuvers (CAM) and the majority of mission-critical operations must be computed and performed autonomously.

For this mission, the idea is to use a standard avionic architecture (PF OBC) coupled with a dedicated payload computer (PL OBC). To reduce the cost and development time needed for the satellite, it has been decided to utilize a pre-existing avionic platform. The first step is to adapt it to the demand of the mission. To preserve the platform integrity and consistency, the team decided to focus on a dedicated payload that has to handle all the remote sensors needed for the rendezvous [101]. Fundamental parts of the software, such as the various image

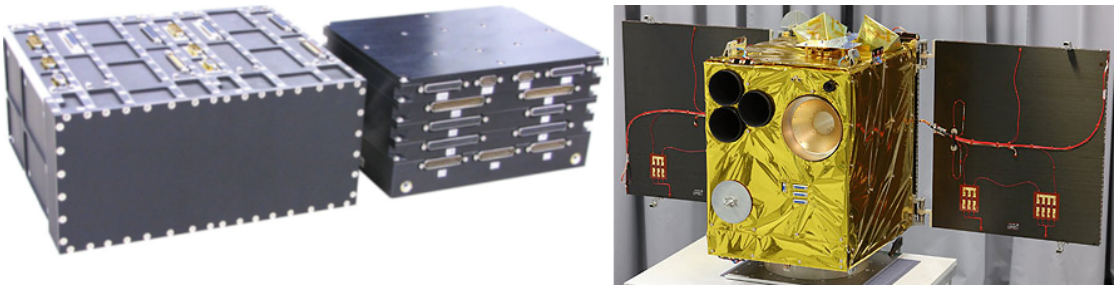


Figure 8 – **Flexible LEO Platform On-Board Computer (OBC) and PCDU[47] & Flying Laptop[47] ©IRS, University of Stuttgart.**

processing algorithms, will be implemented directly on this PL OBC to benefit from dedicated hardware accelerators.

The primary function of the PL OBC is to handle the multiple sensors dedicated to the detection, tracking, and analysis of the target. The Payload On-Board Computer acts as a controller and manages all the modes for the instruments as well as the FDIR. It also has the responsibility to control the data flow from the sensors to the different processing units. The secondary function is to manage the hardware accelerators and ensure the distribution of the data to the correct unit. The last function is to communicate and operate with the main satellite platform (PF OBC) considering the requirements of the precise GNC. It provides processed data to the control algorithms of the spacecraft and reacts to the telecommand sent from the ground. The synchronization between the main platform and the payload core is capital to ensure correct navigation and prevent any collision with the target. Moreover, the payload should be responsive enough to manage all its tasks without causing issues when critical information is requested by the main satellite platform. GNC algorithms for relative navigation remain a typical example where minimizing latency is decisive.

Flexible LEO Platform Testbench

The Flexible LEO Platform (FLP 2)[47, 48] is an avionic platform designed by Airbus DS GmbH, Friedrichshafen and some partnerships. The first version has been launched to space as the "Flying Laptop" satellite from the University of Stuttgart (Figure 8). The satellite was launched in July 2017 in LEO and is still perfectly operational up to this day (May 2022).

The FLP satellite is designed for a $500-800[km]$ orbit with an inclination range between $0-110deg$. It can host a payload up to $100[kg]$ and provides a maximum peak power of $300[W]$ with an orbital average of $200[W]$[47, 48]. In its initial version, the platform contains two separated hardware boxes, one is the Power Control and Distribution Unit (PCDU) and the other is the On-Board Computer (OBC). Collectively, they are designated as the Combined Data and Power Management Infrastructure (CDPI). The PCDU has power management, bus regulation, and few controls over the OBC in case of failure. The OBC hosts the processor, the communication, and an interface board. All parts together, the OBC controls the attitude,

communications, the different sensors, and includes the FDIR. A new version of the platform (FLP 2)[102] is currently being developed in Friedrichshafen. This version will bring higher computation capability, a propulsion module, and a few other features.

In the process of developing the CS-1 mission, it has been decided to use the FLP 2 platform as the main testbench and potential primary avionic for the satellite. The reasons are its modular architecture and its high level of reliability. With the avionic platform, Airbus DS GmbH provides the opportunity to directly develop a payload module with a HIL setup. By obtaining the development board (GR712RC Dual-Core LEON3-FT[49]) with the same processor of the FLP 2 platform and combining it with a simulator developed by Airbus (SimTG), an operational testbench was set up. Its purpose is to reproduce the interaction between the main Platform On-Board Computer and the payload unit with real hardware. In the first iteration, the payload core is a classic board operating a minimal version of Linux that supports various tasks.

To obtain a working environment, the simulator SimTG emulates the entire spacecraft from the different sensors to the communication board and the power distribution. It is a HIL simulation since the main On-Board Computer has the processor as a hardware part (development board). Moreover, it can run the real flight software of FLP 2 since it communicates with SimTG that acts as the rest of the spacecraft. As said before, this setup at EPFL is built to facilitate the design of the PL OBC and minimize development costs. The processor board is kept as a hardware part since it will be the main (and probably unique) interface between the dedicated payload unit and the rest of the platform. Moreover, the PL OBC part needs a tight interaction with the main PF OBC especially to provide relative navigation information during the rendezvous phase. The other advantage of having the true flight software is the ability to test the behavior of the PL OBC as it is developed.

Hardware-In-the-Loop (HIL)

The above testbench at the EPFL Space Center is not something unique. Many instances of HIL can be observed in the literature. C. Kirlin has published an introduction to the concept and some interesting applications in various domain.[103].

By looking at the spacecraft aspect, the case of Attitude Determination and Control System (ADCS) is a recurring theme. Indeed, it often implies the combination of various sensors and dedicated software. Moreover, the ability to test algorithms on the ground is something that is looked at. For example, Corpino and Stesina tested an ADCS and an EPS board for their CubeSat in a HIL setup[104]. They were capable to verify the functional and operational requirements of their satellite. Similarly, Gaber et al. tested the ADCS for their microsat by using a hardware gyroscope board[105]. For them, cost-saving was one of the driving factors. By extension, T. Chang implemented a HIL simulation system for on-orbit docking dynamics[106]. HIL setup for ADCS development can even be found dating back from 1993[107]. In their test, they simulate the entire spacecraft and part of the ground segment to develop the last version of their attitude control.

In a similar domain, Gamazo-Real and Flores tested some COTS electronic components for their microsat payload[108]. They wanted to use the advance of the graphics processing unit (GPU) for data fusion and needed to do some fault injection testing in the electronics for its verification.

A more general look at the Hardware-In-the-Loop testing is done by Himmler et al.[109]. In their paper, they suggest the use of additional software to combine Model-Based Design as well as requirement tracking with the testing. They describe in extensive detail the interconnection between the tools and the advantages of creating such testbench. H. Rumann suggests an alternative approach to HIL setup by focusing on reconfigurable elements with the decoupling of some hardware parts[110]. His reasoning is based on the cost and complexity of developing a one-of-a-kind setup for each spacecraft and pushing instead for multiple design testing.

Related to the satellite world, there is also some interesting research in the Unmanned Aerial Vehicle. Jung and Tsiotras demonstrate the advantage of building a HIL setup to test the autopilot of a small Unmanned Aerial Vehicle (UAV)[111]. They have the flight dynamics simulator connected to their autopilot as well as to a radio-controlled transmitter.

Data bus

The data bus communication protocols represent a decisive aspect of this type of mission. They are especially significant when two separate computers need to communicate in a reliable and efficient way. Many different data buses exist and have been used for space applications with their specific drawbacks and advantages. Initially, an introduction to some concepts is performed and then the protocols used in this thesis are described.

In his book "Onboard Computers, Onboard Software and Satellite Operations: An introduction" [112] J. Eickhoff offers an excellent start to the design and implementation of communication protocol. The MIL-STD-1553B, SpaceWire, and CAN bus are presented. Surprisingly, research in high-speed data buses can even be found in 1988[113]. T. Izumisawa and A. Wakatsuki describe the use of integrated circuits for their bus called CSC (Communications subsystems Supervisory and Control equipment) dedicated to a multi-beam communications satellite. Regarding bus architecture, B. Palminiter et al. explain the use of a linear distributed bus with the I2C protocol for small satellite avionic[114]. They present a hardware/software application with multiple components and show its usage on two satellites. On a similar level, M. Mughal et al. suggest the use of RS-485 for their cubesat[115]. They demonstrate the ability of the bus to be highly modular and emphasize the usage of COTS components. They equally describe in meticulous detail their fault analysis testing. On bus topology, B. Li and Y. Liu show the advantage of using a centralized CAN or UART bus for the implementation of an On-Board Data System[116]. They present in detail the procedure to establish such an avionic architecture in a small satellite class. Ultimately, L. Zhou and J. An suggest the use of an adaptative payload data bus with the support of multiple communication protocol[117]. Their vision is to develop a payload data handling subsystem able to communicate with a

range of various instruments based on a central design.

The five different protocols used in this thesis are described below. For each of them, a brief introduction is provided with some practical cases when they have been implemented in a space mission. The description additionally provides some information relative to the Open Systems Interconnection model (OSI model)[118].

The first protocol implemented in this work has been Remote Memory Access Protocol (RMAP) with SpaceWire network standard by the European Space Agency. It is defined in the "ECSS-E-ST-50-51C – SpaceWire protocol identification"[119] which covers the layer 1 and 2 of the OSI model (physical and data-link layers). The network and transport layer (No. 3 & 4) are defined in the "ECSS-E-ST-50-12C Rev.1 – SpaceWire – Links, nodes, routers and networks"[120]. Regarding the RMAP protocol, it is defined in the "ECSS-E-ST-50-52C – SpaceWire – Remote memory access protocol"[121] and provides description up to the application layer (No. 5) of the OSI model. Spacewire is a full-duplex serial network mainly used in spacecraft avionics. The standard has been designed since the beginning for space application with the inherent constraint associated with it. It uses two differential lines in each direction called data and strobe to send information. Both of them are using Low Voltage Differential Signaling (LVDS). A typical format contains the destination address, a data part that has no limitation in size, and the End-Of-Packet (EOP). As stated, this protocol and network are primarily used by satellites as well as rovers. To cite a few, the Sentinel one to three as well as the number five are using it [122]. On the NASA side, there are two of the major telescopes with JWST and SWIFT[123, 124]. Ultimately, ESA-JAXA BepiColobo orbiter is additionally using SpaceWire[125].

Moving to other data buses, the second one described is the SPI defined by Motorola. Its description covers the physical and data-link layers (No. 1 & 2) of the OSI model. It is frequently employed in embedded electronics to connect processors or micro-controllers with external devices. These devices can be sensors, DAC or ADC, or even memories. This bus is a full-duplex and synchronous interface that has a master-slave architecture. The data are transmitted simultaneously on two lines (MOSI & MISO) that are respectively used by the master and the slaves. The synchronization is done thanks to the rising or falling edge of the clock generated by the master (SCLK). The last line (CS) can be used to select the slave to which the microcontroller is talking. The bus can also be configured regarding the clock polarity and phase. This configuration should be matched on all devices connected to the link. Its usage in spacecraft is widespread but it is rare to witness it as the main data bus. This protocol can primarily be found for specific elements in CubeSat such as UWE-2[126] or KySat-1[127].

The I2C[128] by Philips Semiconductor (now NXP Semiconductors) is also a widely used communication bus. The OSI model layers 1 and 2 are also covered by its specifications. It is a half-duplex synchronous bus that supports multi-master multi-slave architecture. The bus is composed of uniquely two wires. The first one (SDA) represents the Serial Data line that transmits the information in both directions. The Serial Clock (SCL) line is generated by the master and synchronizes the sampling rate of the data. The I2C interface supports in

general lower clock frequency than the SPI bus. In addition, each slave is given an address to determine if a message is sent it. One of the differences with the SPI bus is that the packets of I2C have a defined structure with a limited amount of information included. This bus can be seen in the Myriade family[129] from the CNES as well as in SwissCube[130].

The next protocol that has been looked at is the CAN bus. The ISO11898 standard[131] defines the data-link and part of the physical layer of the OSI model. It is one of the most used communication standards in the automotive industry and has found its way to other domains. The bus allows for multiple users to connect to it without the need for a master. The bus is full-duplex differential and employs two lines to transmit data. Both CAN-High and CAN-Low lines have an identical recessive voltage of 2.5 $[V]$ and they are pull-up/pull-down when a 0 is encoded. The protocol supports two types of frames with the base format and the extended format. The second one accepts 29 identifier bits instead of the regular 11. Both of these frames can contain up to 8 $[Bytes]$ of data. This bus has been used by Surrey Satellites Ltd. in their GIOVE-A[132] (first prototype of the European Navigation System Galileo) as well as in the SMART-1 mission by ESA[133].

The RS-422 (or TIA/EIA-422) is a full-duplex differential data transmission bus and can be considered as an upgraded version of the RS-233. The unique OSI model layer covered by its standard[134] is the first one, the physical layer. It is additionally relatively employed in the embedded world to connect multiple devices to a microprocessor. Nevertheless, the bus uniquely supports one driver line but can include multiple receivers. It includes two sets of differential lines dedicated to the transmission and reception of packets. The primary advantage remains its ability to send and receive data at the same time. It is also more resistant to noise thanks to the differential line. Some instances of the bus are observed in the STP-S26 spacecraft[135] and the UNIFORM-1 CubeSat[136].

The list presented above introduces the main protocol/data bus implemented during this thesis. Nevertheless, many others exist and have been seen in space applications. MIL-STD-1553B, Slink, ASCS and RS-232/RS-485 are other significant candidates that could have been looked at. The reason for not employing them has been primarily their lack of availability on the main development board used during this work.

B.3 Simulation & Optimization

Avionic Simulator

With the rise of complexity in electronics, many companies and research teams have created/used various simulators to help both the development and the verification of their architecture. It is peculiarly valid for complex electronic avionics such as airplanes, drones, and cars where optimization and reliability are essential. In an attempt to create a similar tool for ADR missions, a few simulators for avionic airplanes and autonomous UAVs have been investigated.

J. S. Dittrich and E. Johnson have published in early 2000 the results of their testbed for UAVs [137]. They highlighted the need for a tool to help the design and testing of hardware and software. Their initial step was to develop the navigation software in parallel with the simulation. They used sensor emulators with error models to simulate realistic results. In their approach, the goal was to verify the validity of the algorithm and sensor fusion. They additionally utilized a tool to simulate the UAV in a real environment. Their following step was to create Software-In-the-Loop and Hardware-In-the-Loop simulation to pursue their validation. The attractive part remains in its initial phase with the implementation of sensor emulation. Even though they focused more on the sensor fusion and algorithm performances, their methodology is worth mentioning.

H. Charara and C. Fraboul described a simulation model for an Avionic Full Duplex Switched Ethernet network (AFDX) [138]. With this model, they had to deal with the congestion of frames and delay in the network. Their tool enables both to verify and test various situations and elaborate realistic designs. In a similar domain, X. Li and H. Xiong have published equivalent results on AFDX models to validate timing constraints and sharing resources [139]. With their model, they were capable to evaluate the end-to-end response of such systems. Both of these publications provide fundamental knowledge regarding the complexity of avionic systems and the difficulty to manage the timing of various flows of information.

H. Salzwedel and two other authors explained in two different publications [140, 141] the importance of performance and mission study over functional-level design. They emphasized the criticality of having standard assumptions and criteria among teams on large-scale projects. To reach the performance requirements of a design, they privileged the creation of high-level abstraction models with discrete events and a final state machine. These models allow to guarantee the specifications of the project among the team and speed the development time. Moreover, the validation and verification will be generally simplified. In the second publication, they favored the use of software tools for system design. Such tools can prevent miscommunication between teams working on the same project and minimize significant errors in the overall design. To successfully achieve this, each team has to develop an accurate model of their project with the interconnection to other parts. According to them, the initial model-based design reduces uncertainty on the project before its implementation and development. An exhaustive description of the methodology is provided in the Ph.D. thesis of N. Fisher[142]. Overall, this work shows the importance of early model creation and the utility of system simulation.

Finally, J. Eickhoff et al. developed back in early 2000 some modules for the TINA Systems (Timeline Assistants) [143] to help the generation of timelines for satellite mission planning. The tool focuses on Earth observation mission that requires a large amount of planning with multiple payloads or instruments. Their tool allows the optimization of tasks among multiple orbits to satisfy the requirements imposed by the user. It will output a timeline with the usage of various instruments in the satellite and when they need to be turned on. This tool also includes the actual position of the satellite in orbit to have the instruments observing the

correct location on Earth. The idea of task allocation and optimization is similar to the tool developed in this thesis.

Mixed-Integer NonLinear Program (MINLP)

A Mixed-Integer NonLinear Program[144, 145] represents a type of optimization problem that has continuous and discrete states. It additionally includes nonlinear functions in the objective and/or in the dynamics of the problem. It is described using state and input variables with the second one being the control of the system. The state variables can represent either a physical element like the electrical energy consumption or an abstract one with the mode in which a sensor is. In both cases, they can be written mathematically. This flexibility enables the description of complex problems but at the cost of solving difficulties.

By definition, a MINLP problem contains an objective function that needs to be minimized or maximized. This function remains the core of the design and has to be defined correctly to be feasible by the solver. It is frequently represented as a combination of state and input variables with some weights associated with them. The dynamics of the problem are the mathematical expression of the behavior. It links each state variable in time by defining their first derivatives. They typically include the inputs as well other states. The solver employs the definition of the derivative to compute the value of each variable at the subsequent iteration.

Ultimately, the constraints of the system limit the variation of the state and input variables. For both cases, it is required that they stay inside a certain range. It could be linked to a physical limit for the system such as the maximum data rate that is received at a memory. To be noted that constraints exist for both continuous and discrete variables.

The idea of employing this type of optimization problem in the avionic and space world is, of course, not new. In 1997, J.A. Debardelaben et al.[146] applied this methodology to cost modeling for embedded system design. The core idea was to employ the solver to define an electronic architecture considering some price parameters and the requirement of the project. They claim it results in a cost and quality improvement of the product. K. Nagalakshmi and N. Gomathi suggest the use of MINLP for task scheduling on multicore avionics[147]. It implies regrouping various tasks with different priorities and assigning them to the correct processor core to optimize the execution speed of the avionic. Similarly, M. Schlueter et al. utilized an MINLP to design a multi-stage launch vehicle and some other project[148]. The principle is to develop a model of a launch vehicle and try to optimize the remaining fuel. It considers various constraints like the number of strap-on boosters or maximum dynamic pressure.

Mixed-Integer Quadratically Constrained Quadratic Program (MIQCQP)

A Mixed-Integer Quadratically Constrained Quadratic Program[149] represents a variation of MINLP in the definition. It is beforehand a linear problem which is not the case in the other type. Moreover, the derivative definition of each function is provided for a MINLP in

order for the solver to compute the evolution of all variables. In the case of MIQCQP, it is unneeded since the problem uniquely contains linear variables. For this type of program, the "QC" implies that some constraints of the problem are quadratically linked. It signifies variables are multiplied against each other or among themselves. Ultimately, the latter term "QP" means the objective or cost function additionally includes some variable multiplication.

The general definition of a MIQCQP is presented below. There is an objective function that can include both a quadratic and a linear part. Next, the bound constraints are here to limit the variables in the range of value they can reach. The linear constraints are expressed in a standard algebraic form and link variables to various constants. Ultimately, there is the quadratic constraint where variables are multiplied among each other with the potential addition of some linear terms. Broadly speaking, this classic definition allows the implementation of the entire model. The downside comes with the size of the various matrices needed for a complex system. In the following definition, x represents the variables and the rest are the constants.

Objectives :

$$maximize\ x^T Qx + q^T x \tag{1}$$

Constraints:

$$l \leqslant x \leqslant u\ \ (bound\ constraints) \tag{2a}$$

$$Ax = b\ \ (linear\ constraints) \tag{2b}$$

$$x^T Q_i x + q_i^T x \leqslant b_i\ \ (quadratic\ constraints) \tag{2c}$$

One trick needed for the model is the implementation of the time variation. By definition, each variable needs to have an instance at every timestep of the simulation. It means they are defined N times if N is the number of iterations. Even though it is a convenient way of implementation, the downside comes with the number of variables needed.

MIQCQP is already used in various domains for optimization. For example, Franco et al.[150] employ it to solve a distribution system expansion planning. Their models include multiple types of elements like substations, circuits, or capacitor banks that need to construct or modify. The task is primarily to optimize the number of nodes in their distribution system, but similitude can be found in the optimization approach. In a similar publication[151], a MIQCQP model is used to optimize a water network. The task is similar to finding a trade-off between water distribution and the cost of infrastructure. To limit the issue with the high dimensionality, a stepwise strategy is implemented.

C Thesis Objectives and Main Contribution

The avionic for Active Debris Removal satellite represents a crucial point that is being investigated by many. The constraints and the variety of operations required by the spacecraft make

it a complex problem to resolve. Moreover, the trade-off between autonomy and absolute control imposes many different limitations. Overall, design decisions have to be made to accommodate the need of a specific mission. To support these choices, a deep knowledge of avionics is required as well as a good system engineering overview.

The objective of this research has been to investigate payload avionic architectures for Active Debris Removal missions. Focus has been placed on optimizing the number of components and their arrangement as well as testing protocols for electronic board interconnection.

To achieve these goals, the following steps have been taken:

- Develop preliminary concepts for PL OBC-PF OBC interactions.
- Test various communication data buses and their efficiency.
- Develop a design and optimization tool for payload avionic architecture.
- Analyze the efficiency of the tool's results.
- Verify part of the tool implementation.
- Test the tool with a real case scenario.

The key contributions of this thesis are:

- The development of data encoding and validation method tailored to Active Debris Removal mission. The emphasis has been placed on the efficiency of the data transaction and reliability.
- The implementation of RS-422, CAN, I2C, and SPI data buses into a hardware testbench and analyses of their performances under various constraints.
- The development of a payload avionic simulator which includes the interaction between sensors, main processor, and algorithms. The simulation outputs the behavior of various components over time.
- The mathematical definition of the payload avionic architecture for a MINLP/MIQCQP solver. Particular care has been taken to enable the testing of multiple cases and a broad range of parameters.
- The extensive analyses of design parameters on the optimizer tool. The influence of implementation decisions in the tool is the decisive aspect explored.
- The development of a payload software prototype for the optimizer verification in a dedicated Hardware-In-the-Loop testbench.
- The creation of a realistic payload avionic architecture model for the optimizer.

The results of this research have led to the publication of two peer-reviewed conference papers, three conference papers and one peer-reviewed journal paper:

1. The first development of the avionic simulator has been presented at the 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC) in San Diego and received the best of track award[152].

2. The preliminary study of the hardware testbench as well as the concepts for the Payload On-Board Computer was displayed (virtually) at the 34th Proceedings of the AIAA/USU Conference on Small Satellites (2020).[101]

3. The initial optimizer tool with the MINLP approach was presented (virtually) during the 2021 IEEE Aerospace Conference.[153].

4. The MIQCQP optimizer tool with its parametric analysis was presented (virtually) at the 35th Proceedings of the AIAA/USU Conference on Small Satellites (2021)[154].

5. The validation of the optimizer and the concepts for the future steps have been presented at the 2021 IEEE Aerospace Conference in March 2022 in Big Sky, Montana[155].

6. The extensive study of communication protocols and the implementation of data encapsulation for ADR mission has been submitted to Acta Astronautica in early 2022[156].

D Structure of the Thesis

This thesis is divided into two core parts respectively designated as "Avionic Platform and Testbench for Evaluation" and "Design Simulation and Optimization". They reflect the two paths followed during these four years to advance the development of the ClearSpace-1 avionic.

In the first part, the idea is to present some of the work done on the avionic testbench located at the EPFL Space Center. In chapter 1, some preliminary developments on the testbench are presented with a protocol for the interaction between the main rendezvous payload computer (PL OBC) and the Platform On-Board Computer (PF OBC). The second chapter explores in more detail the usage of various data buses as potential backup or fall-back and their implementation. Emphasis is placed on data encoding, error resilience, and information verification.

The second part focuses on the general development of a simulation and optimization tool. To begin with, the third chapter introduces an initial version of a payload avionic simulator for Active Debris Removal mission. It lays the foundation for the physical interaction between the core components of the payload avionic. Chapter 4 contains the basics behind the transformation of the tool into an optimizer. The goal is no longer to display the behavior of the system over time but to find the optimal set of sensors and algorithms for a given mission.

The fifth chapter is the study of various parameters in the optimizer. It employs parametric analyses to establish the effect of multiple design points on the tool's results. The sixth chapter contains an introduction to the verification of the optimizer. It is the first link between the hardware setup and the optimizer developed. Chapter 7 presents the creation of a realistic payload architecture model for the tool and the analysis of the results.

Part I
Avionic Platform and Testbench for Evaluation

1 Preliminary Concepts for PF OBC - PL OBC

1 Introduction

This chapter is based on the publication "Dedicated On-Board Computer for Active Debris Removal Mission" presented virtually at the 34th Proceedings of the AIAA/USU Conference on Small Satellites 2020[101]. The content has been reviewed to match the structure of the thesis. This chapter describes the first steps toward the implementation of a dedicated rendezvous Payload On-Board Computer (PL OBC) for the Active Debris Removal (ADR) mission ClearSpace-1 (CS-1). The mission will use a standard satellite platform (or bus) developed for Earth observation and combine it with another Platform On-Board Computer (PF OBC) for all the tasks specific to the project. The current testbench setup has the satellite physical processor board connected through SpaceWire and Ethernet to a simulator and a payload computer prototype. By implementing a Hardware-In-the-Loop (HIL) setup, it is possible to assess various configurations for the spacecraft. This chapter focuses on the interaction between the Platform On-Board Computer and the Payload On-Board Computer. The preliminary design of the payload is presented, a communication protocol is established, and a time management method is developed.

For this CS-1 mission, the idea is to use a standard avionic architecture coupled with a dedicated payload computer (PL OBC). To reduce the cost and development time needed for the satellite, it has been decided to use a pre-existing avionic platform. The essential part is to adapt it for the requirement of the mission. To preserve the integrity and consistency of the used platform, the team decided to focus on a dedicated payload. This element will have to handle all the remote sensors needed for the rendezvous. Fundamental parts of the software, such as the various image processing algorithms, will be implemented directly on this PL OBC to benefit from dedicated hardware accelerators. The PF OBC with the physical processor board is also connected to a simulator (developed by Airbus DS GmbH). The simulator (SimTG) can emulate instances of all the satellite's standard components like power and telecommunication even though it has not been used in this chapter. In the first iteration, the PL OBC is a classic electronic board running a minimal version of Linux that handles various

tasks.

The payload's primary function is to handle the multiple sensors dedicated to the detection, tracking, and analysis of the target. The dedicated payload computer acts as a controller and manages all the modes for the instruments as well as the Failure Detection, Isolation, and Recovery (FDIR). In addition, it has the responsibility to control the data flow from the sensors to the different processing units. The secondary function is to manage the hardware accelerators linked to the payload and ensure the distribution of the data to the appropriate unit. The ultimate function is to communicate and operate with the main satellite platform (PF OBC) considering the requirements of the precise GNC. It provides processed data to the control algorithms of the spacecraft and reacts to the telecommand sent from the ground. The synchronization between the main platform and the PL OBC is capital to ensure correct navigation and prevent any collision with the target. Moreover, the payload should be responsive enough to manage all its tasks without causing issues when significant information is requested by the main satellite platform. GNC algorithms for relative navigation remain a typical example where minimizing latency is decisive. The interaction between the PL OBC and the PF OBC remains the primary topic of this chapter.

2 Current Architecture

As stated in the introduction, the current avionic architecture is composed of a standard platform (FLP 2) acting as the PF OBC connected to a dedicated payload computer (PL OBC). The standard platform has the PF OBC as its core and its predecessor (FLP) has been used as an Earth observation satellite that is still in orbit in 2022. It has been designed to be flexible and modular with all the nominal functions of satellite implemented, ranging from the AOCS to the communication protocol. This platform enables the ClearSpace-1 mission team to dedicate their time and resources toward the specific payload computer and the capture mechanism.

This main avionic platform is connected to the dedicated payload computer via SpaceWire and Ethernet. In FLP 2, SpaceWire is the regular network connecting the various boards of the avionic. However, to ease the implementation of the payload, it has the possibility to be linked with an Ethernet cable.

Regarding the software architecture, the FLP 2 platform is mainly coded in C++ with the operating system based on a Real-Time Operating System (RTOS), Real-Time Executive for Multiprocessor Systems (RTEMS). More information on the platform architecture can be found in "A Combined Data and Power Management Infrastructure" and "The FLP Microsatellite Platform - Flight Operations Manual" by Eickhoff, Jens et al. [47, 48].

2.1 Hardware

The testbench setup hosted at the EPFL Space Center is a simplified replica of the one found at Airbus DS GmbH, Friedrichshafen. It has the processor board connected to a SpaceWire router and then an Ethernet-SpaceWire Bridge. The Ethernet connection is then plugged into a regular router connecting to the satellite's simulator (SimTG) and the prototype payload. Figure 1.1 shows the layout at the time of the publication (2020).

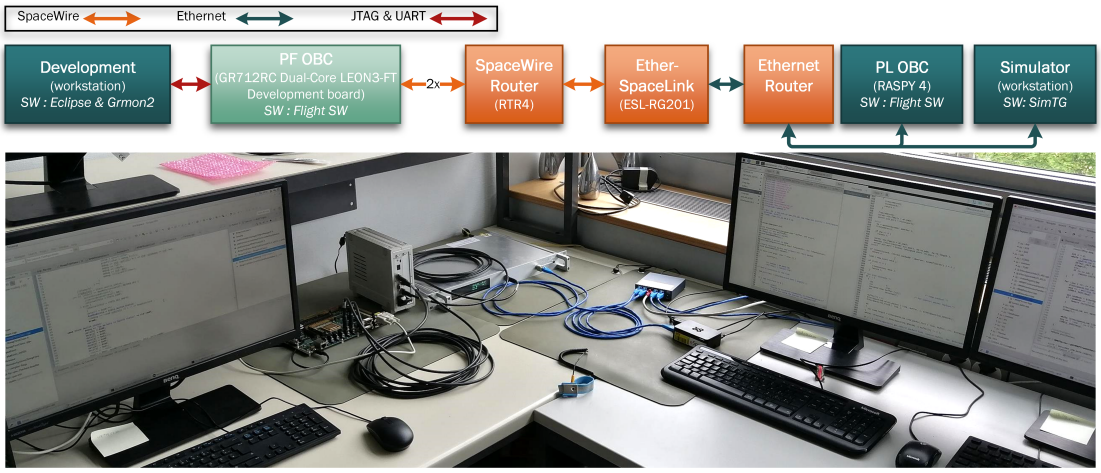


Figure 1.1 – **Current setup.**

The setup additionally included a development workstation connected with UART and JTAG to the processor board (GR712RC) for development purposes. Furthermore, connected to the Ethernet router is the mission control workstation (Not shown in Figure 1.1) which is not being used at the moment but will enable subsequent work.

The following paragraphs are dedicated to the main hardware parts regarding the current state of the project. The first one is the processor board hosting the flight software, then there is the payload prototype, and finally the SpaceWire router and bridge.

Processor Board

For this setup, the PF OBC uses the GR712RC development board built by Cobham Gaisler AB[49] as its main processor. The board contains a Dual-Core LEON3FT SPARC V8 processor with several input/output ports and other functionalities. As the name states, the processor has a fault-tolerant design and supports also radiation-tolerant technology for space application. According to the specification, it can resist up to 300 $[krad]$ of Total Ionizing Dose (TID), has Single-Event Latch-Up (SEL) immunity, and Single-Event Upset (SEU) tolerance. Figure 1.2 shows the architecture of the GR712RC system-on-chip which is the core of the development board used.

These two LEON3FT processor cores have 16 $[KiB]$ of data and 16 $[KiB]$ of instruction cache.

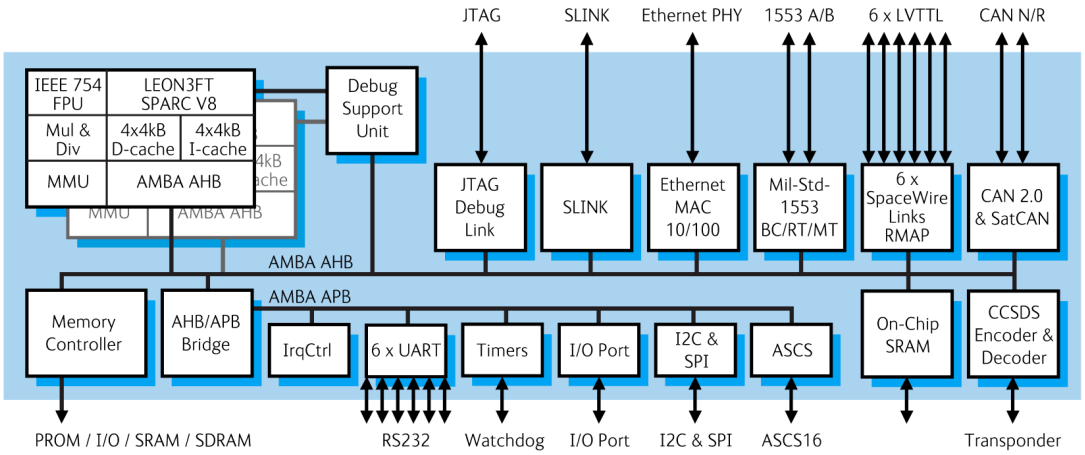


Figure 1.2 – **GR712RC block diagram [49] ©Cobham Gaisler AB.**

They contain two IEEE-754 floating-point units and have each a 192 [KiB] Error Detection And Correction (EDAC) protected on-chip memory. They have various interfaces including 6 SpaceWire links with RMAP target. The processor is allowed to have a maximum clock frequency of 100 [MHz] and the SpaceWire links can go up to 200 [$Mbps$].

In addition, the development board contains an 80 [$Mbit$] Synchronous Random Access Memory (SRAM) and a 64 [$Mbit$] flash memory. It also contains power and clock circuits as well as all the interface connectors for the SpaceWire, CAN bus, Ethernet, JTAG, and UART.

This development board acts as the processor of the PF OBC of the testbench setup. In the following part, it is referred to as processor board or by extension as the PF OBC. Accordingly, the PF OBC would include a lot more elements than just the processor, but it simplifies the explanation to name it directly the Platform On-Board Computer.

Payload Prototype

In the first iteration of the PL OBC, a Raspberry Pi 4 Model B[157] has been chosen to act as the payload processor (IC: 20953-RPI4B). It offers considerable flexibility and a proper range of Input/Output (I/O) ports. This model has Quad-core Cortex-A72 (ARM v8) 64-bit System-on-Chip with a clock frequency of 1.5 [GHz] and 8 [GB] of Synchronous Dynamic Random Access Memory (SDRAM). It additionally includes an Ethernet port and 40 General-Purpose Input/Output (GPIO) pins. Nevertheless, it lacks a RTOS in the first development.

The Ethernet port remains fundamental for this design since it is the way of communication with the PF OBC and the rest of the setup. The GPIO pins will be useful later in the development when prototypes of modules have to be connected to the payload.

Even though the capabilities of this small computer are relatively limited in comparison with the main avionic board, its cost and flexibility allow for rapid development. The primary

purpose at this point is to test the communication between the payload board (PL OBC) and the main processor (PF OBC). Naturally, limitations in the development will rapidly rise, and it will have to be replaced with a more powerful computer. Nevertheless, the communication protocol and the task management can stay consistent.

SpaceWire Router and Bridge

To connect the processor board with the PL OBC as well as the satellite's simulator (SimTG), it is required to use SpaceWire Router and mostly an Ethernet-SpaceWire Bridge. For this setup, the company 4Links[158] is providing both of them.

The router employed is the RTR4 module which contains 4 SpaceWire interfaces. Its primary goal is to route the signal coming from/to the processor board to the Ethernet-SpaceWire Bridge. In the current design of the setup, this module could be omitted since there is uniquely the processor board that needs to be connected with SpaceWire. However, the testbench will evolve, and the possibility to include other hardware elements is significant. One possibility could be to move the payload computer from an Ethernet interface to a SpaceWire one.

The second module is the Ethernet-SpaceWire Bridge called EtherSpaceLink-RG201. This version of the setup contains a single SpaceWire port as well as a single Ethernet port. Its goal is to manage the transfer of packets to/from SpaceWire from/to the Ethernet network. This allows connecting various modules on the Ethernet side without the need for a dedicated SpaceWire interface for each of them. Regarding the SpaceWire network, it uses the Remote Memory Access Protocol (RMAP). For the Ethernet side, the Internet protocol suite (TCP/IP) is required to talk to the bridge, however, it is something handled by most operating systems. This connection is extremely valuable for the satellite's simulator which is a simple workstation and uniquely has Ethernet ports. The second advantage is for the PL OBC that can be a generic small board without any dedicated hardware interfaces for the SpaceWire connector. This module could be updated with a combination of the Diagnostic SpaceWire Interface (DSI) and the Single Link Recorder (SLR) called DSR (Diagnostics SpaceWire Recorder).

2.2 Flight Software

In the following section, the software implemented is a direct heritage of FLP 2 Flight Software (FSW)[47, 48] provided by Airbus DS GmbH, Friedrichshafen to the EPFL Space Center and the start-up ClearSpace. The setup uses this already flight-proven software as a base to construct the code for the handling of the dedicated payload computer. It has to be noted that the PL OBC in itself does not inherit from the FLP 2 Flight Software.

As mentioned, the software consists of two main instances. On one side the modules are directly integrated into the FLP 2 FSW and they need to handle the communication coming from/to the payload. To be specific, there are the data request, data verification, the modes changes of the payload, and the potential High Priority Command (HPC). On the other side, the

PL OBC also needs to receive and send packets to the main PF OBC. In addition, it handles the various sensors connected and allocates the internal or external resources to the algorithms.

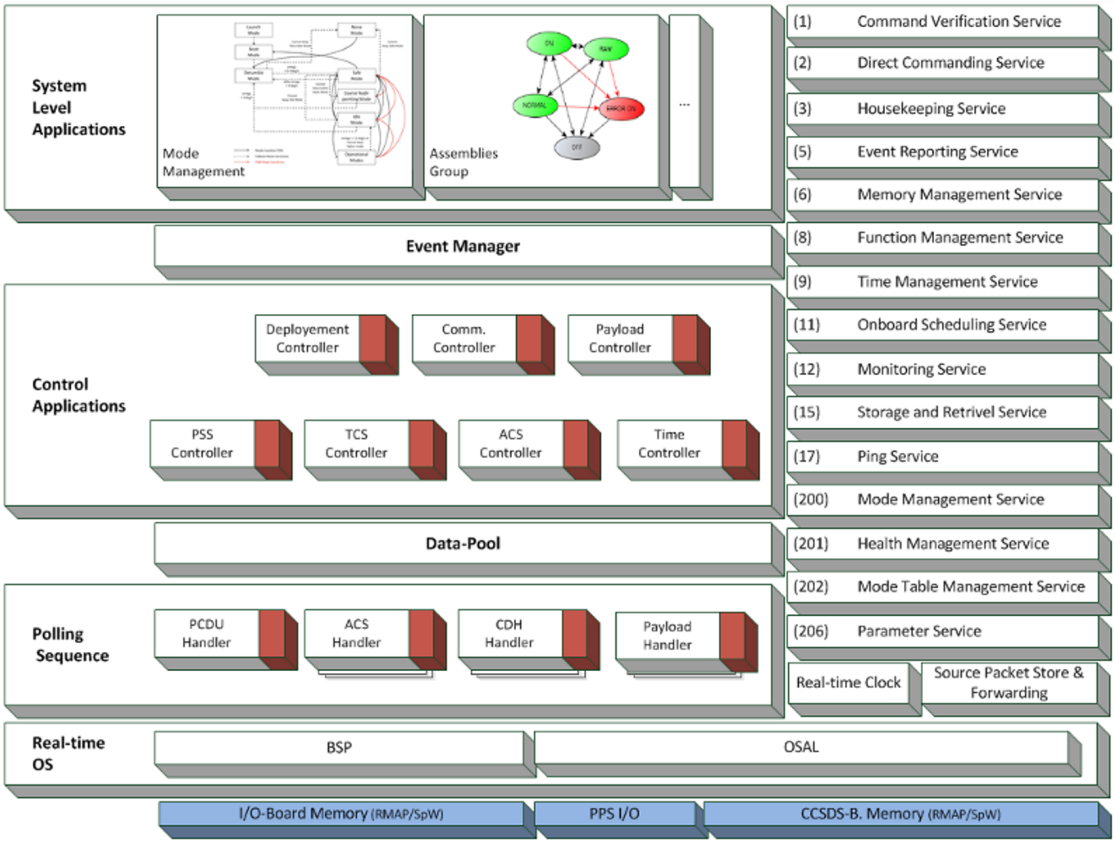


Figure 1.3 – **Flight Software of FLP1 block diagram [47] ©Jens Eickhoff.**

Regarding the FLP 2 FSW, it is based on the Real-Time Executive for Multiprocessor Systems (RTEMS) software which allows using all the functions needed for a Real-Time Operating System (RTOS). The FSW is written in C++ with Object-Oriented Programming (OOP) design in mind. The modules developed for the handling of the dedicated payload follow the same rule.

With OOP, there is a high level of encapsulation and heritage between modules. In the code architecture, virtually all subsystems have a controller instance that manages the element in general. In addition, they contain an assembly module that regroups and commands all the device handlers. These device handlers are directly controlling a single hardware part like a reaction wheel.

Another fundamental concept of the FLP 2 FSW is the datapool (OBSW DP). To save variables that need to be accessible from various places, the software stores them in the datapool. It is physically located in the RAM and contains primarily the state of the spacecraft. It is the location where data produced by the payload computer will be stored.

Regarding the communication protocol, the FLP 2 FSW has implemented the Packet Utilization Standard (PUS) developed by ESA. The PUS services allow a standard way of communication between the ground and the spacecraft. The goal is to extend this implementation to include the need for the dedicated payload, however, it is not planned in the first setup.

The ultimate key point of the software is the implementation of the Failure Detection, Isolation, and Recovery (FDIR) in the code. The topic is not addressed in this chapter and further information about the existing implementation in the avionic can be found in "The FLP Microsatellite Platform - Flight Operations Manual" by Eickhoff, Jens et al. [47].

Figure 1.3 shows the general architecture of the FLP1 Flight Software. In this initial version, the payload was still a module inside the main avionic core. In this diagram, the datapool can be seen and there are also the various handlers.

Concerning the software for the dedicated payload, the goal is to follow a similar type of architecture to benefit from the advantages of FLP 2 platform. Moreover, by utilizing a similar structure of code, it is more straightforward to transfer modules from one architecture to the other and share common modules between the two. As in the first phase of the development, the payload has a few programs developed in C++. It additionally uses standard libraries, especially for TCP/IP sockets management.

3 PF OBC-Payload Interaction

This section is dedicated to the interaction between the dedicated payload computer (PL OBC) and the processor board (PF OBC). As a reminder, this chapter assumes that the processor board of the PF OBC is represented by the GR712RC and the PL OBC is the Raspberry Pi 4. Both hardware boards are connected through SpaceWire and Ethernet as shown in Figure 1.1. At this point, there are no physical sensors or instruments connected to the PL OBC.

In the following subsection, multiple aspects of the interaction between the two OBC are presented with the constraints associated with them. Similar concepts were already developed at Airbus DS GmbH, Friedrichshafen[159] and some have been adapted for the need of the mission.

The principle behind the communication is a Master-Slave architecture with the main avionic core (or PF OBC) being the Master. Since the PF OBC handles all the primary functions of the satellite and most importantly the FDIR, it is key that it remains constantly fully operational. Moreover, the master includes the task to manage the telecommunication with the ground as well as handling a potential High Priority Command (HPC) in case of emergency or failure. In addition, this architecture remains a preferred option to maintain the integrity of the FSW developed by Airbus DS GmbH, Friedrichshafen. Nonetheless, the dedicated payload computer (PL OBC) still has the critical task of providing accurate relative navigation data when in close proximity operation with the target. All the algorithms and sensors dedicated to

the rendezvous phase are handled by the payload, and the information computed in it should be transmitted to the main avionic core. Since the PF OBC handles the AOCS and propulsion part, it requires the information of the PL OBC to compute the maneuvers and react according to the state and attitude of the targeted debris.

With the PL OBC working as a slave in the architecture, all the interactions are handled by the PF OBC. By consequence, the platform has to create a request RMAP packet every time information is needed. The protocol follows this sequence. If the PF OBC needs information from the slave, it creates a write request packet and sent the message to the payload. The PL OBC receives the message and transmits directly a small acknowledge (ACK) packet. The payload accesses the information required in its own memory/datapool and creates a reply message. In parallel, the PF OBC starts sending read request packets to the payload in a periodic manner after receiving the previously acknowledged packet. (It is designated as the polling sequence.) These messages are acquired by the slave and the read reply information is sent to the PF OBC as soon as available. The PF OBC receives the read reply packet and analyses the correctness of the information. If the data is valid, it sends an acknowledge message otherwise a new write request is created. Figure 1.4 shows the protocol in a simplified way. The following subsection addresses the various requirements and details of the interaction.

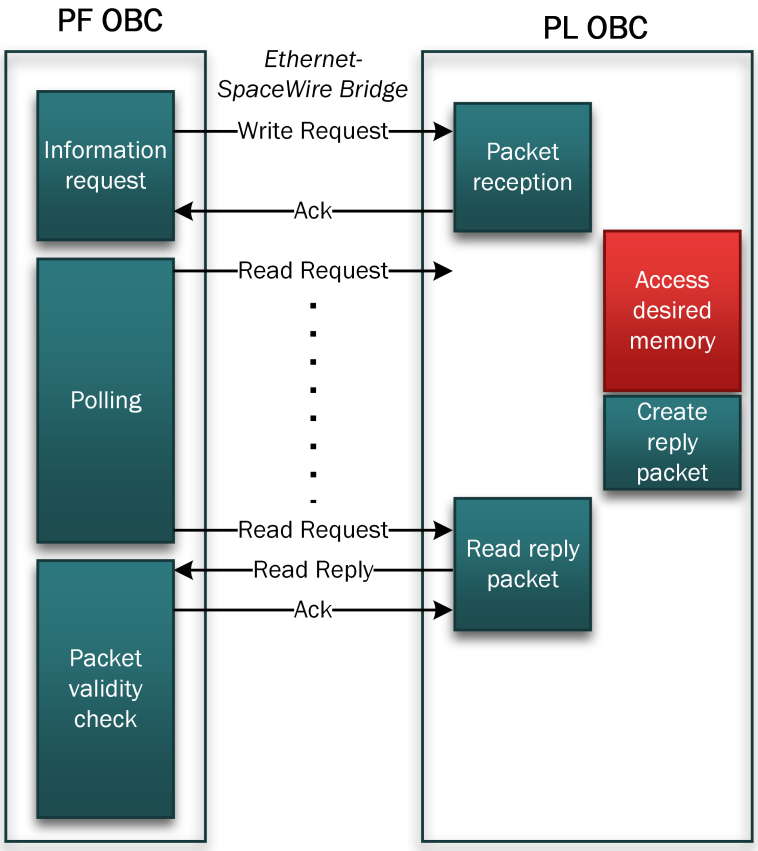


Figure 1.4 – **Communication protocol.**

3.1 Packet Transformation SpaceWire to Ethernet

This part is specific to the usage of the PL OBC connected with Ethernet to the PF OBC. As shown in Figure 1.1, the PL OBC is connected with Ethernet and then SpaceWire cable to the PF OBC through an Ethernet-SpaceWire Bridge. In this case, the role of the bridge is to correctly redirect packets coming from one side or the other. However, the encapsulation of these packets is done inside both modules.

From the SpaceWire side, all the packets are transmitted using the Remote Memory Access Protocol (RMAP). The decision is directly inherited from the FLP 2 FSW architecture that already uses this protocol internally. To keep the consistency with the rest of the avionic, it is used for communication. Nonetheless, a few additions needed to be done to communicate with the payload computer. Since it is connected with Ethernet to the Ethernet-SpaceWire Bridge, the TCP/IP protocol is employed for the Ethernet part. It implies the PF OBC also needs to handle and control the TCP/IP aspect. To achieve this, the data have to be encapsulated into first a TCP/IP packet and then the RMAP for the SpaceWire part. A schematic of the encapsulation can be seen in Figure 1.5. To be noted that this concept was already explored at Airbus DS GmbH, Friedrichshafen[159]. As shown in the figure, the TCP/IP headers are added in front of the data packet and then encapsulated inside an RMAP message. In order for this to work, the PF OBC needs to have prior knowledge of the IP address of the PL OBC.

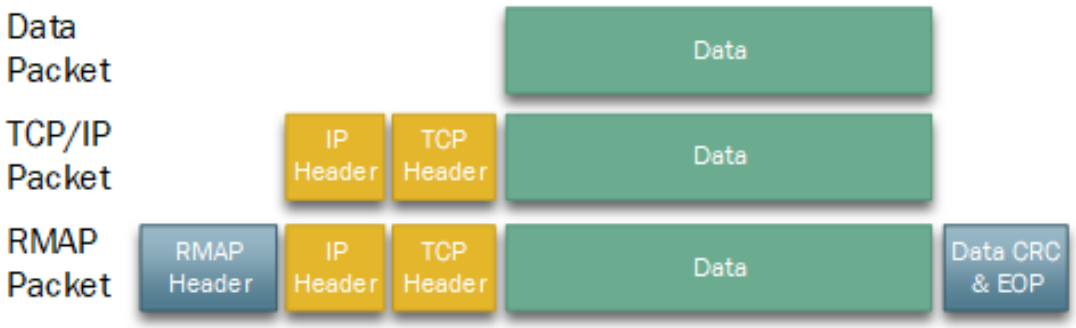


Figure 1.5 – **Packet encapsulation TCP/IP & RMAP.**

When the packet mentioned before is received by the Ethernet-SpaceWire Bridge, the RMAP Header and end of the frame are removed. It is then routed accordingly to the TCP/IP information. Next, the TCP/IP packet is received by the PL OBC and decoded. For the return path, the payload computer does not need to handle RMAP packets. Indeed, the transmitted information requires to be written inside a buffer in the Ethernet-SpaceWire Bridge and then requested by the processor board. It means the PL OBC simply need a TCP/IP connection to the bridge to receive and send information. On average, it simplifies the communication protocol between the two entities.

As a summary and following the Open Systems Interconnection model (OSI model)[118], the Ethernet part follows the Internet protocol suite. In this case, it implies the Transmission Control Protocol (TCP) for the session and transport layer (No. 5 & 6) and the Internet Pro-

tocol (IP) for the network layer (No. 3). Regarding the SpaceWire network standard[119], it technically uniquely covers the physical and data-link layer (No. 1 & 2). Nevertheless, additional standards[120] defines the network and transport layer (No. 3 & 4). Finally, the Remote Memory Access Protocol (RMAP)[121] includes additional definition up to the application layer (No. 5).

One of the disadvantages of the TCP/IP protocol is the timing issue. It generally ensures the correct delivery of the packet, but the time of transmission can vary. Nonetheless, this protocol could allow multiple payloads to be connected on the same network and work in parallel. A possible solution to this problem is the restriction to uniquely SpaceWire network connection between the PL OBC and the PF OBC. However, it would require the modification of the PL OBC. A second solution would be to change the transport layer with the User Datagram Protocol (UDP). It has the advantage of general faster transmission respectively to Transmission Control Protocol (TCP) and is generally well suited for real-time applications. The downside is obliviously its reliability with the lack of delivery guarantee. In addition, the Ethernet-SpaceWire Bridge is not able to support this protocol in this implementation.

3.2 Reaction Time of the PL OBC

A problem related to time-sharing is the availability of the computed data in the PL OBC. It has to be remembered that the various instruments required for the rendezvous phase have different frequency rates as well as computation times. It implies that pre-processed data from the sensors are computed at a different rate and then the pose estimation algorithm needs some additional time to compute its result. Even though this timing is mostly deterministic, they need to be considered.

The classical example is the PF OBC (or processor board) requesting position information to the payload. In this case, the processor could require the relative distance as well as the rotation rate of the target with both information coming from different sensors. The request is emitted at a time t_0 and received by the payload at $t_1 = t_0 + \delta t_{transmission}$. At this point, the payload will access its memory and retrieve the needed information. The relative distance has been computed at an earlier time t_2 with the use of the radar sensor data. However, the information is valid at the time of measurement t_3.

$$t_3 = t_2 - \delta t_{distance\ algorithm\ computation\ time} - \delta t_{radar\ sensor\ preprocess} \tag{1.1}$$

Similarly, the rotation rate estimation has been done at time t_4 with measurements from the camera at t_5.

$$t_5 = t_4 - \delta t_{rotation\ rate\ algorithm\ computation\ time} - \delta t_{camera\ sensor\ preprocess} \tag{1.2}$$

These two measurements are encapsulated in a packet and sent to the PF OBC with this timestamp. Ultimately, the processor board receives the information requested at $t_7 = t_6 + \delta t_{transmission}$. Nevertheless, the data in the packet have been computed at respectively t_2 and t_4 with time $t_7 - t_2 \neq t_7 - t_4$. At this point, it is apparent that the PF OBC needs information relative to t_2, t_3 and t_4, t_5 to correctly compute its desired maneuver. For this reason, it is capital that data packets include the timestamp at which they have been created, the time of measurement as well as the time of computation end. Figure 1.6 is a summary of the previous explanation. It shows the timeline with the various timings as presented in the example.

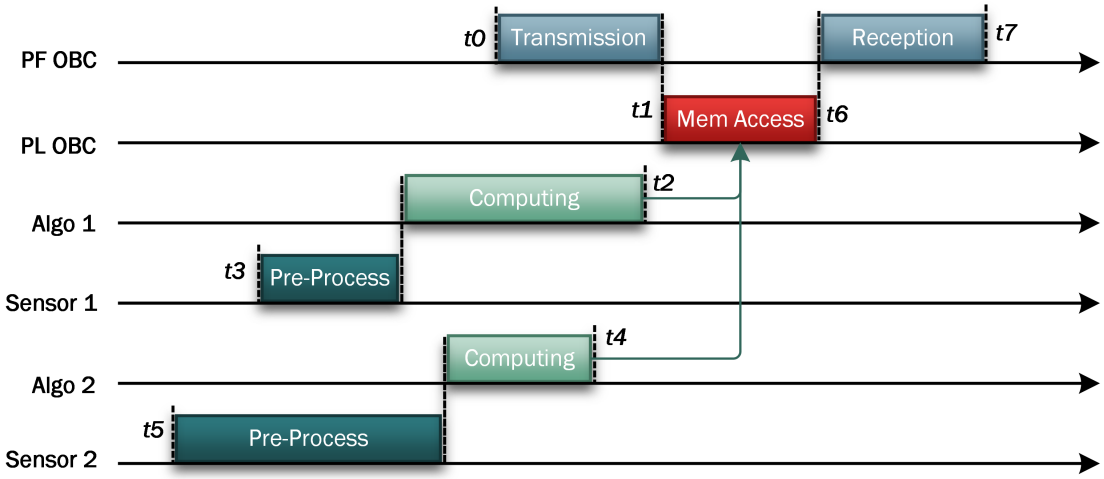


Figure 1.6 – **Timing of measurements and transmission.**

3.3 Time Synchronization Method PF OBC-PL OBC

One of the most constraining challenges of the PF OBC-PL OBC communication is the timing or synchronization. It is crucial for both parts to work autonomously, however, the exchange of information required precise timing. A typical example of this is the handling of a critical phase like the last meter of the rendezvous with the targeted debris. In this concluding phase, the spacecraft requires an elevated level of control regarding its attitude and will correct any disturbance with its AOCS. This subsystem is of course handled by the PF OBC. Nonetheless, the pose estimation and relative attitude between the chaser and the target are provided by the PL OBC. Indeed, all the rendezvous sensors are directly controlled by this computer. Moreover, the data retrieved are employed by algorithms held in some parts of the payload computer. Overall, the PF OBC needs information stored in the PL OBC to compute its exact attitude in space relative to the target. Since the distances during the concluding phase are small, any disturbances can be fatal and thus the attitude computation should be as precise as possible. It means that the main avionic platform should know exactly when the last pose estimation of the target has been done to deduce its own location in space and execute the proper maneuver. From this, it is obvious timing is decisive. The previous subsection has explained the importance of correct timestamps for the measurement. This subsection is

about the synchronization of the payload and PF OBC.

In the configuration shown in Figure 1.1, there is no direct circuit linking the PF OBC to PL OBC. Everything is transmitted through SpaceWire and Ethernet. A possibility would be to have a dedicated clock circuit that can share time information through the whole spacecraft, but it is not implemented in this work.

For this setup, the most straightforward method is purely to include timing information inside the packets that are being transmitted. In this approach, it would imply that each packet that transits between the PF OBC and the PL OBC has a timestamp incorporated inside. The timing information is handled predominantly by the processor board to correctly update its current position and attitude in space. As explained before, the AOCS needs this information to compute the exact position relative to the target at any moment in time. Once updated, the spacecraft can propagate its attitude with a Kalman filter or similar algorithm to deduce its current position. Figure 1.7 shows the construction of a data packet with multiple pieces of information in it and is based on the example of Figure 1.6. In this example, there is a general timestamp for the packet as well as one for every measurement and computed value.

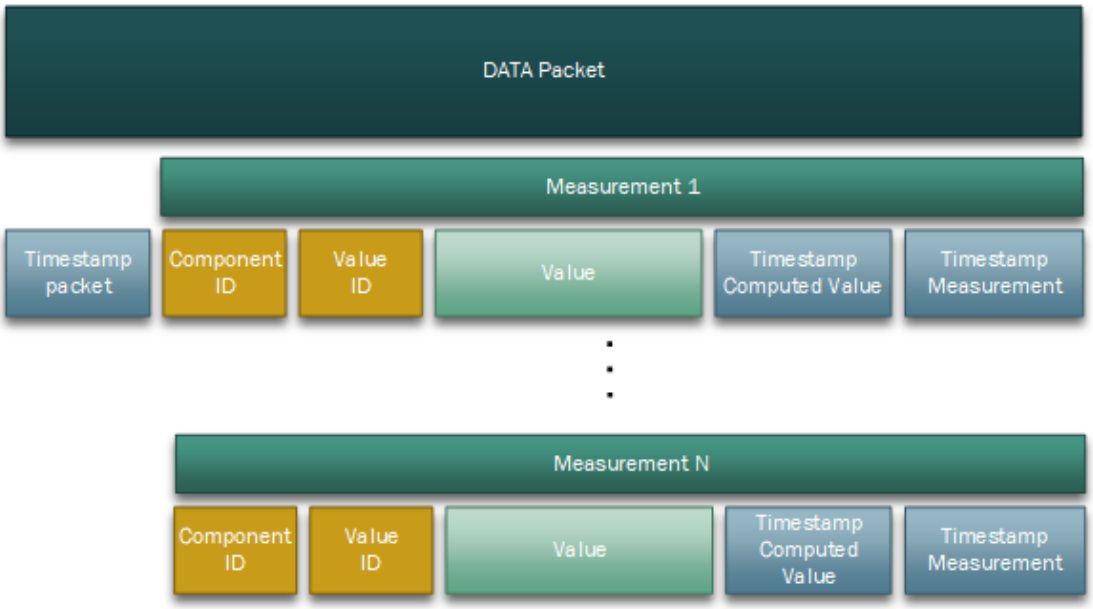


Figure 1.7 – **Structure of data packet with multiple measurements.**

In this configuration, it implies both entities (PF OBC and PL OBC) need to maintain access to their respective clock information and the point is not about synchronization but readability. Another problem that needs to be tackled is the precision of the clock. As mentioned in the previous part, it is capital to consider the proper timing for all the measurements to prevent errors in the pose estimation. Slight mistakes in the attitude computation could lead to erroneous maneuvers and jeopardize the mission. To overcome this problem, the clock information needs to be shared at microsecond precision and therefore increasing the size of

the packet transmitted.

The ultimate point of this section remains the synchronization of clocks. To minimize the error across the system, it is crucial that the whole spacecraft shares a common clock. A solution is to use a Pulse Per Second (PPS) time synchronization protocol with an additional connection. The basis is to generate a signal with a frequency of generally 1 $[Hz]$ and connect multiple devices to it. It allows for all the devices to be synchronized and prevent time drifting. Obviously, such protocol does not transmit information about the exact time and date. For this reason, the implementation of at least two clocks is needed. There is the central one for the PF OBC module and another one for the PL OBC. An important point to notice is that the clock is commonly generated by the main platform and more precisely the GPS instrument. Indeed, because of the nature of such components, they have access to the exact time and date. If for some reason the GPS module is turned off, then the processor board will generally handle the PPS time synchronization protocol. Since the mentioned protocol does not carry exact timing, a periodic packet is sent through the regular data bus from the processor board to the PL OBC with this information. It contains the exact current timestamp to correct any issue. For the correctness of this update, it is decisive to have a determined transmission time for the data. Since all the messages are sent over the same data bus, the processor board must limit the transmission of any other information. It is especially true at the sending time to prevent delay of this particular packet. Nonetheless, it is recognized that accurate timing for TCP/IP packets can be difficult, and thus more investigations need to be done on the topic.

3.4 Handling & Verification of Data in PF OBC

The following step is then to extract the data from the RMAP packet and analyze it. For the second message (read reply), the goal is to check the integrity of the data and the correctness of the message. First, the header and the end of the packet are analyzed to ensure the validity of the packet. If the encapsulation is correct, the PF OBC controls the structure of the data sent and the timestamp of the message. Assuming the structure matches the requested information of the write request packet, then the data correctness can be checked. If one of these four tests returns an error, the PF OBC will start again its communication protocol with the payload.

Regarding the data correctness, the two main objectives are to verify that each measurement is in an appropriate range and that the timing is plausible. The first check is controlling the ID of the components and values that match the hardware part of the payload. At that point, it is the verification related to the timing of the measurements and computed values. The timing of all information is checked against a defined table and if the difference is too substantial, an error is created. In this case, the communication protocol is reset. The values are then compared with a range table stored in the memory and corresponding to the actual phase of the mission. If the measurement received differs too considerably from the previous one or if it is not plausible to obtain such value, then the communication protocol is started again. At this point, it is worth mentioning the two previous cases can occur because of an issue with

the PL OBC itself or one of the instruments. Nevertheless, the case is not discussed in this subsection and more information on the verification can be found later in the chapter.

Once all the checks have been done, the obtained data can finally be stored in the datapool of the FLP 2 FSW. A summary of the operation is shown in Figure 1.8. It shows the process flow of the decision and the various errors that can be generated.

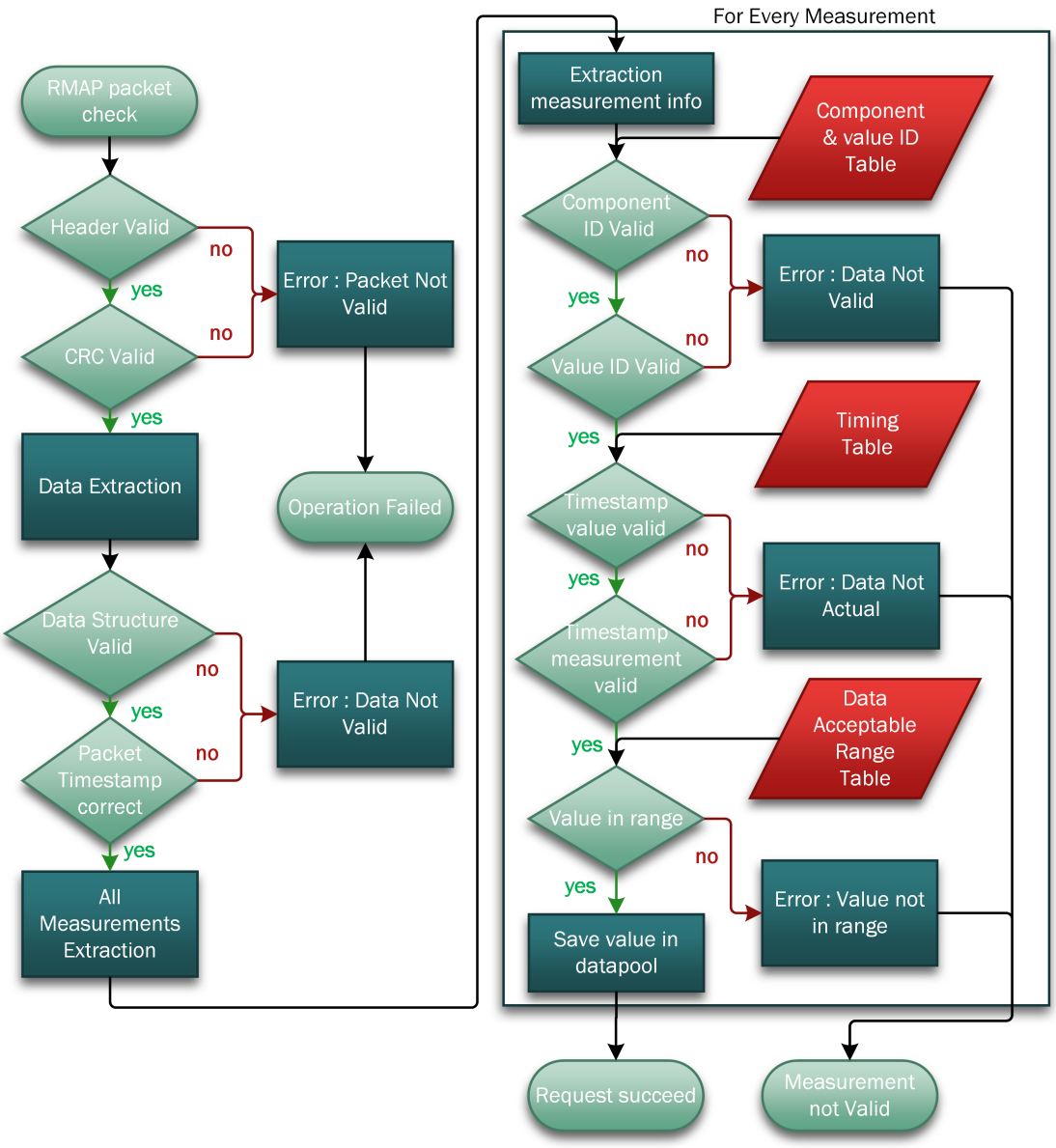


Figure 1.8 – **Validation of read reply packet in the PF OBC.**

3.5 Task Simulation on Payload

At this initial stage of the setup development, the list of sensors that are needed by the spacecraft during the rendezvous phase is still not entirely defined. The requirement regarding the measurement and the type of information has been established but the selection of these instruments is in process. In the same category, the various algorithms needed for the pose estimation and the GNC are being developed in parallel by the start-up and some partners. Moreover, the implementation of the algorithm in the dedicated payload is being discussed and various options are considered to optimize electrical energy consumption, accuracy, timing constraint, and development cost.

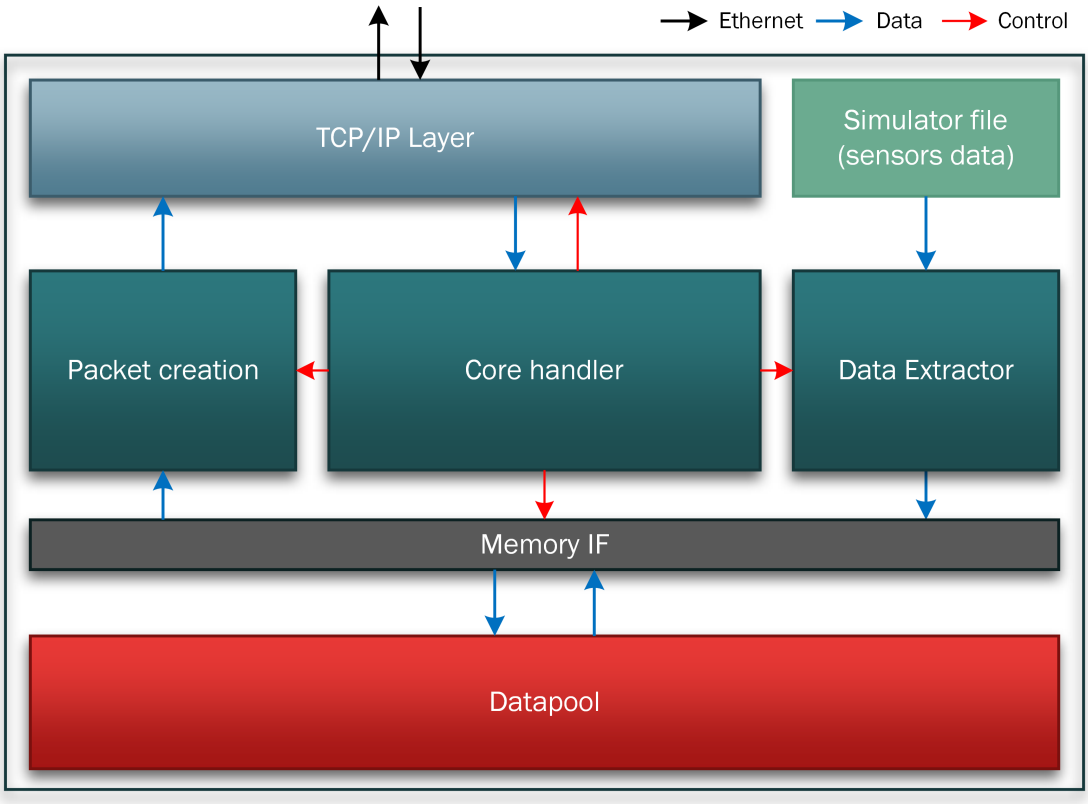


Figure 1.9 – **Block diagram of the Payload On-Board Computer architecture.**

As it will be revealed in the second part of the thesis, analyses of the various architecture configurations for the payload have been done. These chapters show the advantages and disadvantages of diverse payload architecture and the effect of multiple parameters on the system. Based on the result, it is possible to create a simulation of the task in this dedicated payload.

For the first iteration of the setup, the key point is to simulate the sensors' output and the computation time of the algorithm. To simplify the process, a list of outputs is generated prior to the simulation with potential results from the various algorithms with a defined timestamp

associated with them. To maintain consistency, the results are generated in accordance with the scenario of the second part of the thesis. The created list is then polled by the payload computer and accessed as if the sensors and algorithm were producing the data. The information is then stored in the PL OBC datapool and is ready to be requested by the processor board. A block diagram of the prototype payload architecture is shown in Figure 1.9.

This simplified version of the PL OBC allows the development of interaction with the PF OBC. It can be achieved without being dependent on the implementation of the algorithm and the drivers for the sensors. By decoupling the algorithm and the communication part, a reliable transmission protocol can be developed without suffering from changes in the algorithm development or hardware modification.

The downside of this method is the reduction of processing time and resource utilization by the PL OBC. Since no real algorithms are being run by the payload computer, all the processing resources are allocated to the communication and datapool handling part. It implies that timing constraints and latency are not matching the ones of the future implementations. Nevertheless, the hardware used for the payload computer is not representative of the final one either. It means the implementation of more real algorithms in the actual version of the PL OBC will differ from the constraint on the final payload. This trade-off has been decided to accelerate the development of the communication part of the PL OBC and verify the concept of operation.

4 Challenges

This section addresses the various challenges laid in the previously explained processor board and payload interaction. The goal is to present the current issues with the implementation and possible solutions to these problems.

4.1 High Priority Command from PF OBC to Payload

As explained in the previous subsections, the timing between the PF OBC and the PL OBC is decisive. To have the appropriate attitude of the target, the main avionic core must have timestamps associated with measurements as close as possible to reality. Despite this required accuracy, there are few cases where the platform needs information potentially as quickly as possible. One example could be that the previously computed attitude state of the target shows that a collision with the target is imminent. Another possibility would be an unidentified part of the debris is actually loose and could potentially be on a collision course with the spacecraft. Such situations are obviously very rare. In addition, the relative velocity when the spacecraft is a few meters away from the debris is generally very small. It means a sudden collision or event can almost never happen. Another possibility could come from an erroneous computation of the target debris state and thus a quick reaction from the main spacecraft. As for the other cases, such situations are highly improbable, especially considering the amount of validation

and verification of all elements of the mission.

Nevertheless, in a hypothetical situation where such an event is happening, the spacecraft would need to react with a maneuver to leave its intersect trajectory. Nonetheless, it still needs to verify the correctness of the new trajectory and ensure the collision will not happen. For the verification, the PF OBC needs to request again information from the sensors of the payload as quickly as possible. In this case, a regular data request from the PF OBC could be too slow. For this reason, a High Priority Command (HPC) protocol could be envisioned. The goal of the extra protocol is to access in a shorter period of time and with priority to the required information. The following paragraphs describe the possibilities of implementation.

One possibility is the use of the regular SpaceWire and Ethernet connection, however, it imposes already many restrictions on the timing. The work could be performed at the reception and processing of the request by the PL OBC. If the request is handled faster on this computer, a timing amelioration could be achieved. It would imply the modification of the task management by the payload and probably the stop of most processes inside it. For example, the PF OBC could allocate all its resources to the received request while blocking the thread of the other parallel tasks. This could indeed improve the timing reaction of the payload computer, but a majority of the delay is still inside the network and with the encapsulation process.

A second possibility would be to add a direct data line connecting the PF OBC with the payload computer. This solution would indeed considerably reduce the latency by removing most of the communication layer needed when going through SpaceWire and Ethernet. Nonetheless, it will create additional hardware interfaces and the complexity of the information transfer will rise. Moreover, the verification and validation of this architecture will drastically increase in difficulty. In addition, it will cause issues in the FLP 2 FSW integrity and impose additional constraints on the payload side.

Another aspect of the HPC is the type of information request that could be done. Since the timing remains the prime driving factor, the information requested should be as small as possible. Nevertheless, it has to contain the critical data required for the spacecraft. At this point, a more intensive analysis should be performed with the AOCS teams. The task will be to determine the type of information that could be needed by the PF OBC with the HPC.

The verification of the data integrity should not be forgotten. Even though the process needs to be the fastest, it is key as well that the information delivered is correct. By looking back at the example discussed before, the HPC is used when mission-critical events are happening. In the case of the trajectory correction, it is at a similar level of importance that the information provided to the AOCS is the most recent and perfectly accurate. If the HPC cannot guarantee the second point, it could create a potentially fatal scenario for the mission. One might be the wrong estimation of the closure rate to the target and thus the collision with the object.

Ultimately, the implementation of a High Priority Command into the avionic architecture is not likely. Since the probability of a sudden event during the final phase of the capture is close

to zero, the HPC might never be used. In addition, it is possible to mitigate most of the risk by using more robust mission planning. For example, decreasing the relative velocity at the capture and performing additional inspection of the debris from a few meters away would greatly reduce the risk. In most of the cases, the PF OBC would require measurements and computed information with a high level of accuracy and not at a faster rate.

4.2 PF OBC Polling and Redundant Information Transmission

In itself, this subsection partially represents an issue. Regarding the state of the spacecraft and information available to the avionic core, it does not cause a problem that the PF OBC requests updates at a high frequency. Indeed, the data requested might already have its most recent value stored in the datapool. It can be assumed the PF OBC has a proper polling sequence for most of the information generated by the dedicated payload computer. It means its FSW guarantees to have frequent updates on the data computed. This process is particularly effective to check the consistency of the instrument and algorithms on the payload and monitor any changes in the trend of the data. An example is the beginning of the relative navigation of the spacecraft. The chaser switches from an absolute positioning in space with the GPS signal and Star Tracker to a relative positioning with the target. It means the PL OBC starts computing values. By constantly polling the payload computer, the PF OBC can monitor the error due to the initialization of the pose estimation algorithm. In addition, it can check the difficulty to measure precisely the target state at a long distance. The monitoring is essential to prevent any modification of the chaser trajectory based on none accurate measurements of the rendezvous sensors. Furthermore, the PF OBC can monitor if the algorithms are converging to a steady solution or if they need to be reset. It can also request a restart of a sensor if the information seems to be erroneous.

The second aspect remains the resources allocated to the polling during various phases. As shown later in the thesis, the PL OBC is generally busier when the chaser is close to the target. In addition, the PF OBC can experience a similar trend in processing resources utilization. In the concluding phase of the rendezvous, the AOCS is constantly used. In addition, the communication with the ground is at its peak to maintain some human control on the satellite. For these reasons, it is crucial to operate the polling sequence at a reasonable rate to prevent overloading the PF OBC. Nevertheless, the PF OBC should still guarantee a very precise update of the debris' attitude to prevent any collision. The risk is not about unexpected events but computational errors.

In conclusion, the request for redundant information by the PF OBC is not a capital point in the development of the mission. The validity of the data as well as the precise timing should remain the driving factors.

4.3 Integrity Check of Data Packets

One of the challenges of the PF OBC regarding the interaction with the PL OBC is investigating the validity of the information received. As discussed in a previous section, the data should be verified accordingly to its associated timing and the plausibility of its value.

For the first point, the time of measurement could directly be compared with the actual time of the spacecraft. If the information is considered to be too "old", then the PF OBC starts once more the data request protocol. This simple verification can prevent the system to be updated with data from sensors being idle or turned off. Indeed, the PL OBC naturally provides the last available measurement or computed value in its memory and the PF OBC has to examine it. If the processor tries to retrieve information that the payload is no longer updating, it could cause an issue. Since the default mode is to restart the data request protocol if the information is considered not valid, the PF OBC could be blocked in a loop of queries. One solution is to ask the status of all payload components when too many message requests fail. Such health status request is commonly used in this domain and serves this exact purpose. By confirming the availability of the sensors/algorithms where the information is produced, the PF OBC can react accordingly. Naturally, it could be an error that has occurred in the payload and a few components need to be restarted. This is addressed in the following subsection.

The second point to check is if the data received are plausible. It implies more logic due to its complexity since the validity of most measurements depends on the distance to the target. Moreover, few sensors might include a larger uncertainty when reaching the limit of their application. It is essential each mission phase employs the appropriate set of elements in their respective range. In addition, the PF OBC should possess a previous knowledge of what value to expect from which sensor depending on the mission phase. A table with the range of expected values will have to be computed prior to the operation on a simulator and then stored in the PF OBC memory. Nonetheless, the possibility to update the table after the launch should be provided. Assuming the avionic core possesses such a list, the check could be done by requesting the modes of the payload and its algorithms/sensors. The following step is to match them with the database. As in the previous case, the PF OBC will start once more the data request protocol if the information does not fit in the predicted range. Similar behavior for the timing validity process should be applied in case of multiple failures.

4.4 Failure Detection, Isolation, and Recovery

This topic has already been addressed briefly earlier. Importantly, this subsection provides uniquely a short introduction to the FDIR concept at the PL OBC. Even though it does not cover this specific payload aspect, more information about the existing FDIR in the FLP 2 can be found in the book “A Combined Data and Power Management Infrastructure”[48].

The main avionic core carries out the fundamental role to control and monitor the PL OBC. It is as well essential for the payload computer to do the same with all the sensors and hardware

accelerators if any. The implication is the full avionic system needs to have two separate FDIR entities with their respective action and procedure and they should be able to communicate. Moreover, it must be remembered that the power control is done by the Power Control and Distribution Unit (PCDU) of the FLP 2 platform and the payload does not communicate with this module directly.

As envisioned, the FDIR of the payload computer will likely have limited action. For example, the integrity check of the data is handled by the PF OBC and the powering of the module should be requested as well by the processor board. Nonetheless, the FDIR can still maintain a range of actions. Regarding the detection and isolation, the payload computer can already monitor the status of the various instruments by requesting their housekeeping data. For the recovery part, software reset is initiated by the payload computer on its own or from an PF OBC request. The "hard" reset should be initiated by the PCDU on demand from the PF OBC by turning off the power of the targeted component.

5 Outcome & Incoming Challenges

This topic requires some subsequent development to better adapt to the need of the missions. The first step is to verify the interaction protocol established in the setup. Various functions have been developed in both the FLP 2 FSW and the payload software and interaction between them should be controlled. By accessing the memory outputs of both modules during all scenarios testing, it is possible to ensure the correct transmission of information. This step is addressed in the following chapter.

The second step is to verify the various timing constraints of the data exchange protocol. As explained in the previous part, it is crucial to monitor the time required by a packet to be transmitted back and forth. Moreover, preliminary assessments can be performed regarding the memory access of the payload as well as data integrity checks in the PF OBC. The following chapter contains some analyses of the timing.

The third step is the implementation of the concept presented in the challenges section. These concepts first need to be refined and improved. Once clearly established, the procedure developed can be incorporated into the FSW. Following their implementation, early verification should be done.

The following steps are not enumerated since they can happen in any order. Among them, there is the amelioration of the FDIR concept for both the payload and the PF OBC. The one presented previously is an introduction to a solution and needs to be extended. Moreover, a deeper analysis of the system needs to be performed to reveal the various cases of failure and the recovery methods associated with them. In addition, future development should follow a more agile process, especially if multiple people are involved. It will smooth the transition to Product and Quality Assurance (PA/QA) phase.

One significant part of the setup will be the implementation of the ground control loop. This interaction already exists for the standard FLP 2 testbench setup and it needs to be modified to take into consideration the payload. The interfaces of the ground control will be modified, and the communication protocol with the spacecraft updated. Once these actions are performed, an operator will be capable of commanding the payload computer through the PF OBC.

In parallel with these implementations, the upgrade of the PL OBC should be considered. As explained in the introduction, the raspberry pi represents a preliminary prototype of the payload to ease the first development of the setup. By upgrading to a more realistic prototype, the possibility to verify timing constraints will be possible. Moreover, the software could be adapted to the new hardware, and the allocation of resources will be more realistic.

In an identical line, a new payload computer will imply the incorporation of the various algorithms needed for the rendezvous phase. In the first phase, they will be fed by pre-computed data but the possibility of adding true sensors is not so far behind. Adding realistic hardware parts will permit the verification of the overall loop as well as the transmission protocol between payload and PF OBC.

The communication management between the PF OBC and the dedicated payload computer is a fundamental point for the future avionic system of the mission. It is a key aspect that can lead to many issues if not implemented in a precise manner. As shown multiple times, the main PF OBC is extremely dependent on the results of the PL OBC for mission-critical parts of the rendezvous. The information computed by the payload computer should be retrieved with high accuracy and in a periodic manner by the main avionic core. It allows the AOCS and other subsystems to work properly.

In this chapter, the concept of interaction between the PF OBC and the payload has been presented with emphasis on the timing of the measurements and the clock synchronization. A validation procedure has been established and a data packet format created. Moreover, the potential implementation of High Priority Command (HPC) and the Failure Detection, Isolation, and Recovery (FDIR) has been discussed. The following step is the verification of the various concepts in the Hardware-In-the-Loop setup with resilience as the primary driving factor.

2 Backup Data Bus for PF OBC - Payload Interaction

1 Introduction

This chapter is based on an ongoing journal paper "Active Debris Removal Mission: Backup Data Bus for OBC - Payload Computer Interaction" that has been submitted to Acta Astronautica in early 2022[156]. This chapter describes the implementation and testing of multiple communication data buses for the interaction between the Platform On-Board Computer (PF OBC) and the Payload On-Board Computer (PL OBC). It is tailored for an Active Debris Removal (ADR) mission like the ClearSpace-1 (CS-1) satellite. The primary goal is to operate a standard avionic platform for satellite and connect it to a dedicated payload computer. This second hardware has the key task to handle the rendezvous and capturing part. In this chapter, the emphasis is placed on the data exchange between the two boards with an Hardware-In-the-Loop (HIL) setup. Multiple communication data buses are implemented, and efficient data encoding is tested.

As explained before, the avionic of CS-1 is built around a standard satellite platform and a dedicated payload computer. This choice has been made to mitigate the cost and development time of such a satellite. It means the primary effort is toward the PL OBC and how the data are exchanged between the two. Indeed, the payload computer remains the vital point for the rendezvous sensors and the image processing algorithms. It implies a large amount of data streamed to the board and powerful hardware to be able to withstand the load of the algorithms. Nevertheless, without communication to the PF OBC, the payload can not accomplish its purpose. The satellite requires the information to navigate and proceed with its routine task.

To approach this problem of data exchange, the critical point is to establish a Hardware-In-the-Loop setup where various communication data buses can be tested easily. This work directly pursues what has been established in the previous chapter. The significant aspect is to offer an alternative in case of the main link failure. Even though it is assumed that the main data link is fully redundant and has different layers of protection, a potential solution is to create a second independent communication channel to support it. The aim is to offer a

fall-back option for the satellite with the ability to transfer decisive information between the two satellite computers. By analogy, the case of the OBC AP-101[160] for the Space Shuttle is interesting. Indeed, this module was using four computers running in synchronization with an additional backup one with a different software. The purpose of this addition was to mitigate risks in case of software errors in the main ones. Evidently, this extra hardware was desired to ensure the safety of the astronauts in the Space Shuttle. In the case of the fall-back communication channel, such efforts could be interesting regarding the current space debris situation. Since any collision would have enormous consequences, a fall-back option might represent a small effort to mitigate the risk. This study can also be useful for a cheaper and easier alternative to implement instead of the SpaceWire/Ethernet bus presented in the previous chapter. It can be used for ADR missions targeting small debris and where using a less complex communication channel could be beneficial.

The starting point of this chapter is to develop a flexible software for both the PF OBC and the PL OBC. It includes the shared high-level procedure for the communication and the data methods. Simultaneously, it is capable to employ the various drivers for the multiple data buses and adapt to their specificity. In this implementation, four different data buses are tested: I2C, SPI, CAN, and RS-422. Some comparison is done with a SpaceWire/RMAP bus, but it has not been tested in this chapter. To support the data buses mentioned earlier, some complement modules have to be used by the payload. In this setup, the PF OBC has a Real-Time Executive for Multiprocessor Systems (RTEMS) operating system. On the other side, the PL OBC currently operates with a minimal version of Linux. This point is discussed in greater detail later.

2 Current Hardware Architecture

The current testbench setup used at the EPFL Space Center is primarily focusing on the interaction between the Platform On-Board Computer and the Payload On-Board Computer. The platform processor board is connected with multiple interfaces to the payload. The primary and initial connection is with a SpaceWire/Ethernet network. The PF OBC is connected through SpaceWire to a router and then a bridge. The bridge has an Ethernet connection to an ordinary router and the PL OBC is connected to this element. The connection for the other data buses uses direct lines between the main development board and the PL OBC. In all cases, the lines used were between 10 and 20 [cm] long. They may not represent the cables' length in the actual flight avionic. To be noticed that the payload needs some special hardware HAT for the CAN and RS-422 data buses. Figure 2.1 shows the general connectivity in the testbench.

The setup additionally included a development workstation connected with UART and JTAG to the platform processor board for development purposes.

The goal of this testbench is to test part of the avionic architecture of a satellite such as CS-1. Figure 2.2 shows the typical simplified layout with the Platform On-Board Computer and the Payload On-Board Computer. It is important to notice that this diagram does not include all

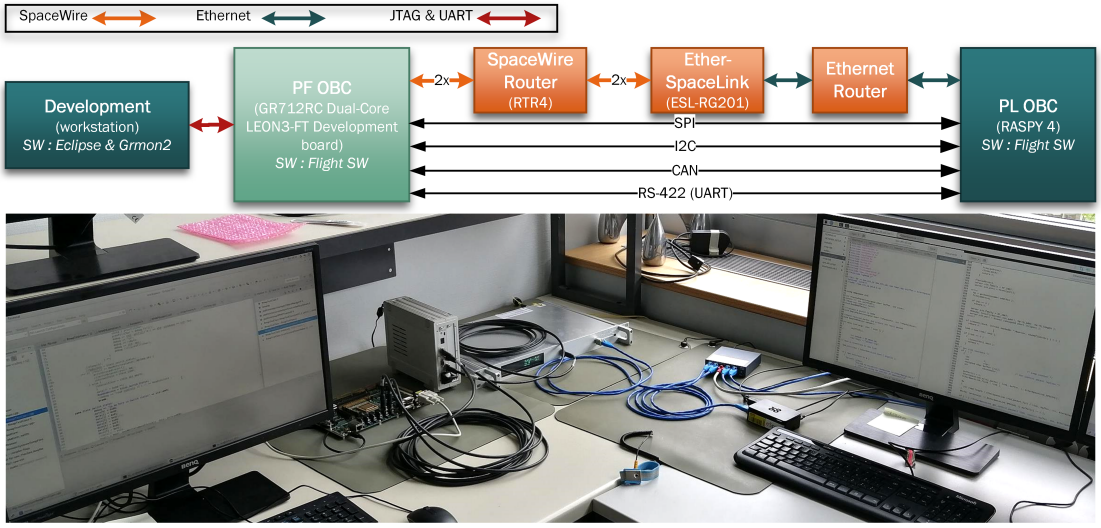


Figure 2.1 – **Current setup for data bus testing.**

the satellite regular elements inside the main avionic platform. It has the AOCS subsystem and the GNC algorithms only for references. Moreover, the sensors dedicated for the rendezvous are simply indicated as two generic gray boxes. The key aspect of the HIL setup is to test the interaction between the two On-Board Computers.

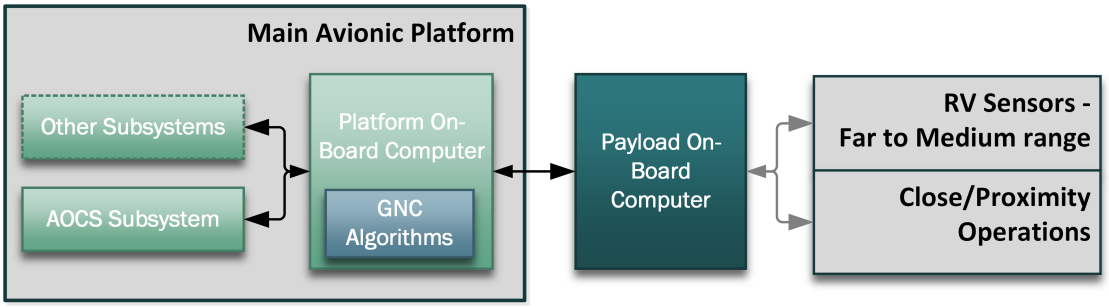


Figure 2.2 – **High-level block diagram of the tested avionic.**

The following paragraphs are dedicated to the main hardware parts in the current state of the project. The first one is the platform processor board which hosts the flight software, then there is the payload prototype board, and finally the extra module for the PL OBC.

2.1 Platform Processor Board

For this setup, the CPU of the PF OBC is represented by the GR712RC development board built by Cobham Gaisler AB[49]. It contains a Dual-Core LEON3FT SPARC V8 processor with several input/output ports and other functionalities. As the name states, the processor has a fault-tolerant design and supports also radiation-tolerant technology for space application. According to the specification, it can resist up to 300 $[krad]$ of TID, has SEL immunity, and

SEU tolerance. Figure 2.3 shows again the architecture of the development board.

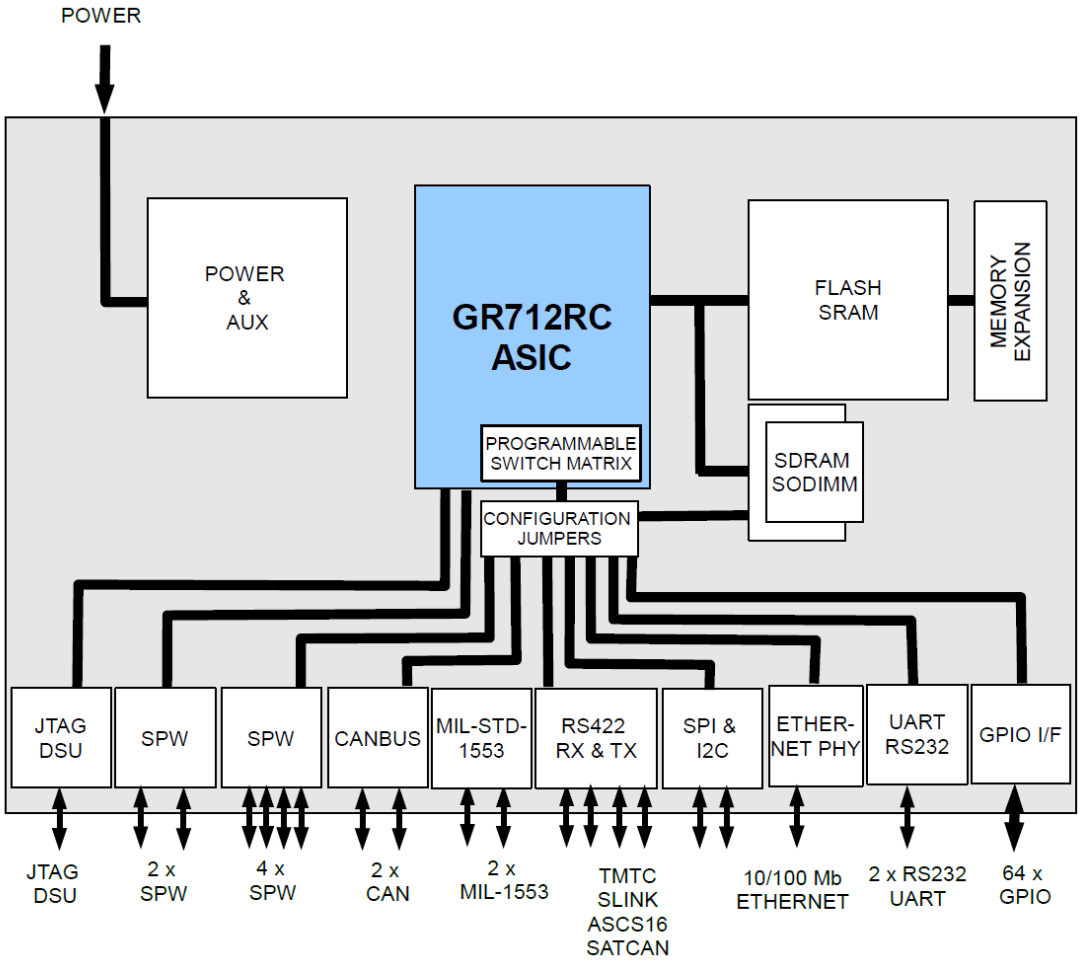


Figure 2.3 – **GR712RC block diagram [49] ©Cobham Gaisler AB.**

These two LEON3FT processor cores have 16 $[KiB]$ of data and 16 $[KiB]$ of instruction cache. They contain two IEEE-754 floating-point units and have each a 192 $[KiB]$ EDAC protected on-chip memory. They have various interfaces including 6 SpaceWire links with RMAP target. The processor is allowed to have a maximum clock frequency of 100 $[MHz]$ and the SpaceWire links can go up to 200 $[Mbps]$. As mentioned earlier, it is running with the Real-Time Executive for Multiprocessor Systems (RTEMS) operating system.

In addition, the development board contains an 80 $[Mbit]$ SRAM and a 64 $[Mbit]$ flash memory. It also contains power and clock circuits as well as all the interface connectors for the SpaceWire, CAN bus, I2C, SPI, Ethernet, JTAG, and UART. The SpaceWire interface uses a DS90LV047A & B for its driver and receiver circuit. The CAN bus has an SN65HVD230 transceiver and the RS-422 a DS34LV87(RS422) transceiver.

This development board acts as the processor of the PF OBC of the testbench setup. In the

following part, it is referred to as the platform processor board or by extension as the PF OBC. Accordingly, the PF OBC would include a lot more elements than just the processor, but it simplifies the explanation to name it directly the Platform On-Board Computer.

2.2 Payload Prototype

For this development, a Raspberry Pi 4 Model B [157] has been chosen to act as the PL OBC (IC: 20953-RPI4B). It offers considerable flexibility and a proper range of Input/Output (I/O) ports. This model has Quad-core Cortex-A72 (ARM v8) 64-bit System-on-Chip with a clock frequency of 1.5 [GHz] and 8 [GB] of SDRAM. It additionally includes an Ethernet port and 40 GPIO pins. In this study, the basic Operating System (OS) of the board (Raspberry Pi OS based on Debian) is kept. Evidently, not using RTOS is convenient for development purposes and allows to use standard tools. Nonetheless, this choice has created some issues when running live tests of communication transmission. A switch to such OS should be done if further testing is required with this electronic board.

Its GPIOs are useful since they allow supporting the various data buses. For this work, the native implementations of the SPI and I2C buses are used. These pins are also needed to connect extensions and are used with additional modules described below.

Even though the capabilities of this small computer are relatively limited in comparison with the main avionic board, its cost and flexibility allow for rapid development. The primary purpose at this point is to test the communication between the payload board (PL OBC) and the platform processor (PF OBC). Naturally, limitations in the development will rapidly rise, and it will have to be replaced with a more powerful computer. Nevertheless, the communication protocol and the task management can stay consistent.

2.3 Payload Prototype - Extra Module

In addition to the payload prototype board, two extra "hats/shields" are used. These components are added to enable the use of specific data buses. No extra module is required by the main development board since the GR712RC already supports the needed data buses.

The first one is the "RS485 CAN HAT" and enables the use of RS-485 bus or CAN[161]. For this application, the second one is targeted. To achieve the functionality, it uses an MCP2515 as CAN controller and a SN65HVD230 for the transceiver. This module employs SPI to communicate with the Raspberry Pi.

The second module is the "RS422 / RS485 HAT for Raspberry Pi" that is used for its RS-422 mode full duplex[162]. It contains the "ISO3080DW" chip and employs I2C for interaction with the payload prototype.

3 Software Architecture

This section describes the general behavior of the avionic system as well as the overall communication protocol. The main principle is to develop two mockups of Flight Software to test the exchange of information between the PF OBC and the PL OBC. The software architecture has to be flexible enough to allow the use of multiple communication data buses. For this experimentation, four different ones are tested: I2C, SPI, CAN, and RS-422(UART). As stated before, SpaceWire/RMAP is not tested in this chapter.

The principle is the PL OBC has some data available in its internal memory, and the PF OBC polls them when needed. It is assumed that the PL OBC is linked to a few sensors and has some algorithms running. The information handled by the payload for the platform processor is the one already processed by fictional algorithms. In its current implementation, the PL OBC is directly reading an external file with values at every main iteration of the software. It consistently includes the last measurements processed by the algorithm in its memory, and the data are linked to the id of the element (sensor). In addition to the value itself, the PL OBC stores two timestamps. The first one is when the information has been computed. The second is the timestamp of when the algorithm of the PL OBC has received the initial measurement. Even though this is a reasonably straightforward way to implement the data storage, it establishes a baseline for the testing.

3.1 Platform On-Board Computer Architecture

In this development, the software architecture of the PF OBC has been reduced to the minimum. Since the goal is to test various data buses, the FSW has been clean out of all other processes linked to the PF OBC. In summary, its unique tasks are to request some data, accept the reply, and check the consistency of the information. For this application, the PF OBC acts as the master of the system, and thus it is the one directing the data exchange. An important point to remember is that the GR712RC development board act as the platform processor of the PF OBC, thus the software is implemented in this board.

The initial task of the PF OBC after opening the communication channel to the PL OBC is to send a request. The default mode asks for the information from some of the three different algorithms. (It is an arbitrary design choice to use only three.) These algorithms represent potential processes implemented in the payload computer to determine some orbital parameter of the target with the data from various sensors. The PF OBC sends the ids of the algorithms in which it is interested. In order, the information transmitted to the payload is the timestamp of the request, the number of ids asked (here between one and three), and then the list of algorithm's ids. Additionally, the information includes an identifier to signal it is a request packet. The diagram in Figure 2.4 shows a typical message.

The following step is to wait for the answer of the payload. The exact protocol for this exchange is described later. Once the data are received, the PF OBC needs to decode the packet, check

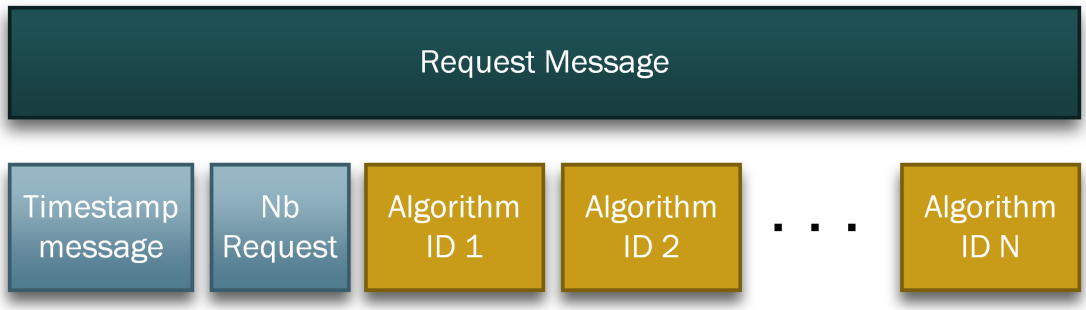


Figure 2.4 – **Structure of request message.**

the validity and store the information. In this version of the software, uniquely the validity of the reply is controlled, and the values are directly displayed on the terminal. Regarding the control, the PF OBC focuses on the amount of information received and checks that each measurement features the exact number of fields. The correct structure of the reply message is explained in the following subsection. Ultimately, the validity check of the reply follows the procedure described in the previous chapter.

3.2 Payload On-Board Computer Architecture

The essential goal of the PL OBC is to possess an updated list of measurements ready to be sent to the PF OBC. In other terms, the payload acts as the salve in the communication. In this implementation, no real algorithm is implemented, and all the values are generated from an external script with dummy information. The first task of the payload is to update regularly its internal memory (or data pool) with the latest information. The second one is to accept the request of the PF OBC and to decode them. The third task is to create a reply message and to transmit it to the PF OBC.

Regarding the first task, the payload is extracting information from an external file at every step of the main iteration loop. It is then updating its internal data pool with the latest values. In the current design, the data pool can uniquely support one type of data which is some values (measurements) computed by fictional algorithms. In its implementation, the data pool always holds the last value of each algorithm. It means every time there is an update, the previous value is erased. This is, of course, not ideal for an actual mission, however, it is preferred to decrease the complexity of the system at this point. As explained earlier, a measurement includes multiple information in it. There is the value that is a floating number. In addition, the id of the algorithm and the id of the value are stored next to it. The first one allows to clearly determine from which source the data comes. At the end, there are two timestamps respectively one for the computed value and one for the received measurement.

The following task is to accept the request from the PF OBC and to decode it. As explained earlier, the request contains the amount of information queried and the list of ids. The

payload simply checks that the number of elements received matches the one requested. Once decoded, the list of measurements is retrieved from the data pool.

The final task is to prepare the reply message and to transmit it to the PF OBC. The structure of the message follows the configuration shown in Figure 2.5. At the beginning, the timestamp of the reply is incorporated, and then the number of measurements sent. For each measurement of the message, there is the algorithm id, the value id, and then the value itself. At the end, the two timestamps are added. The figure shows a generic reply message with N measurements asked

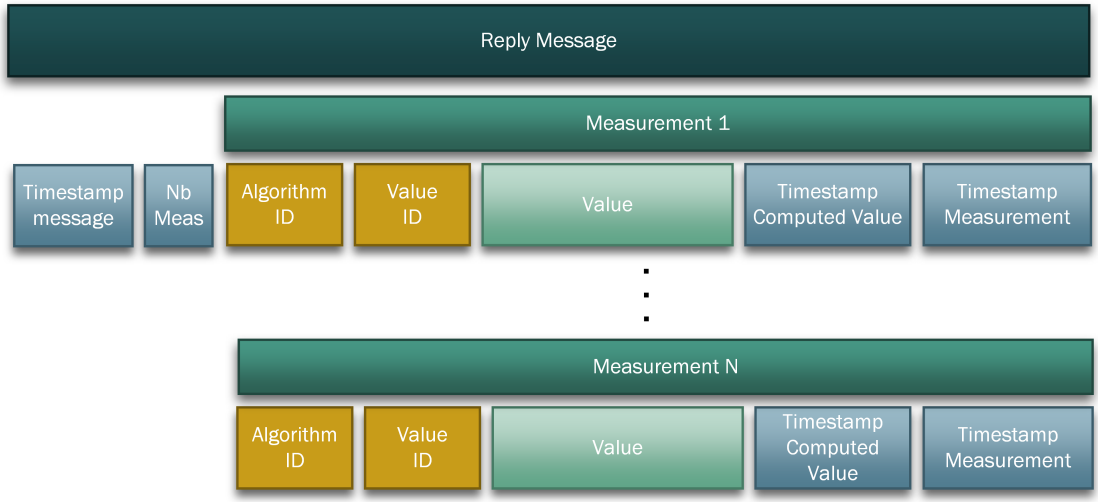


Figure 2.5 – **Structure of reply message.**

3.3 High-Level Data Exchanged

This subsection is about the data exchange between the platform processor (PF OBC) and the payload (PL OBC). At this point, no consideration is undertaken regarding the exact data buses used. The overall data exchange is following the procedure of the previous chapter with some slight modifications. The updated version can be seen in Figure 2.6. This high-level protocol covers the presentation and application layer of the OSI model (No. 6 and 7). The overall procedure is similar to the MIL-STD-1553 data bus standard[163].

Initially, the platform processor sends a request message to the payload. An essential point to mention is that the request can be sent in multiple packets due to limitations in the data bus used. Once the first packet has been received, the payload directly answers with a $wait$ message. If for some reason the packet received does not follow a request format, an error is sent back. This exchange continues until the payload detects the end of the message in the last request. If for some reason the end message is not detected and no new packets are coming, a timer resets the state of the payload.

Once the PF OBC has finished transmitting its request, it initiates its waiting phase. In this

phase, a simple *wait* is sent to the PL OBC. On the other side, the PL OBC prepares the reply message and starts to forward it. As for the request, this message can be split into multiple packets. As long as they are being sent, the PF OBC should reply with a *wait*. If it is not the case, the payload will try resending the same packet. Once all the packets have been transmitted, the payload sends a *stop* to indicate the end of the transmission. At this point, the platform processor can answer an acknowledge message if all the data have been received correctly. If it is not the case, an error is sent and the payload assumes it has to resend the whole reply message.

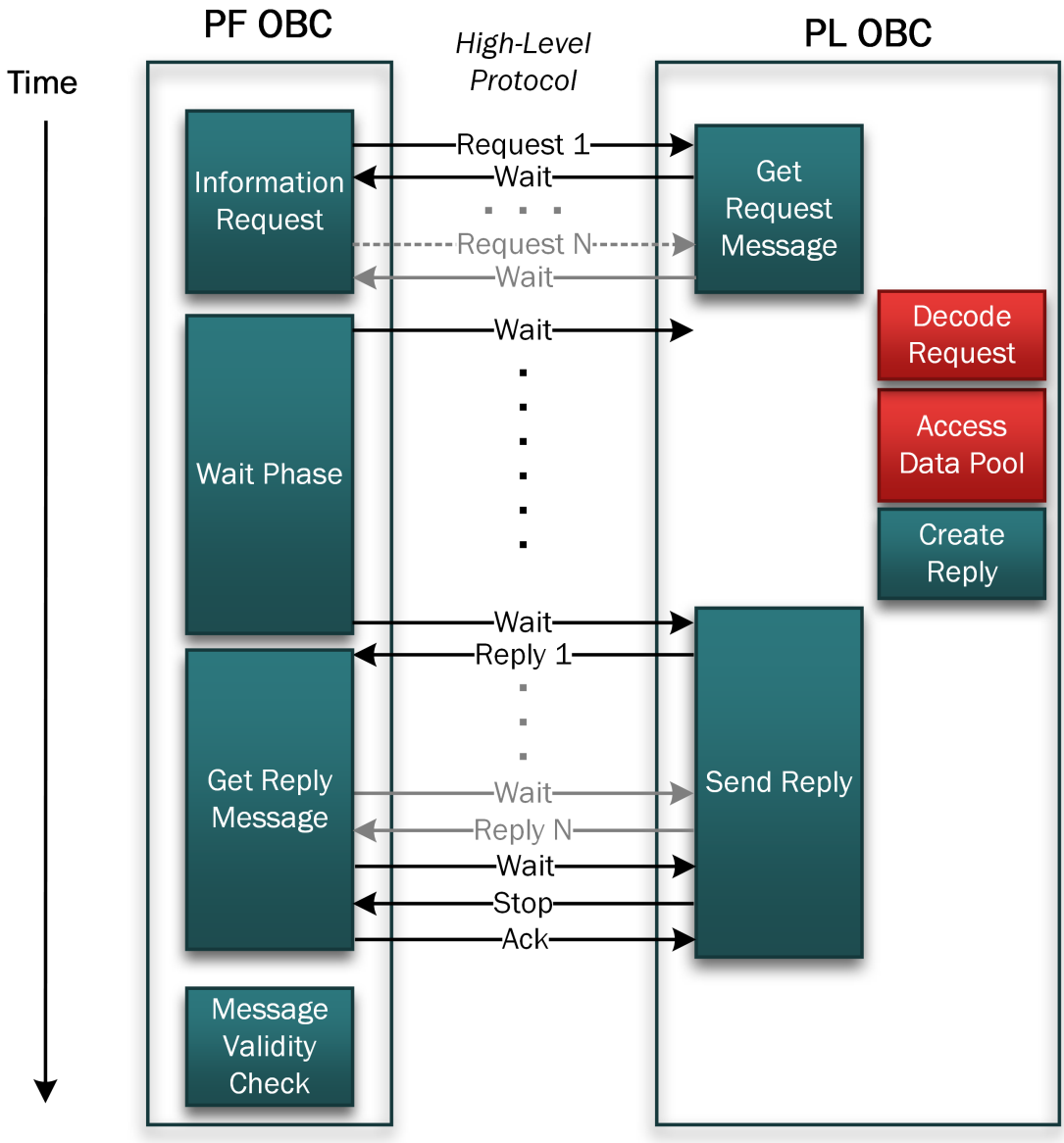


Figure 2.6 – **High-level protocol of the data exchange.**

4 Data Encapsulation and Encoding

One of the significant challenges of this work is to identify the proper way to encode and encapsulate the data to optimize the transmission. This section is not talking about the data buses' specifications and neither regarding the encoding used by specific data buses.

4.1 Keywords

At the software level, it is important to define a few decisive messages to transmit critical information about the state of both the PF OBC and the PL OBC. To keep it minimal, the following short messages are defined:

- $WAIT$ indicates that the sending board is waiting to receive a message. For some data buses, an addition is appended at the end.
- $STOP$ is transmitted by the sender to indicate that all messages have been sent.
- ACK is used to inform the receiving board that all messages have been received.
- ERR informs the receiver that there is an error in the previous message and it should be resent. For some data buses, the message can have further information.

It is crucial to mention that these messages are kept short to increase the speed at which they are transmitted. A fundamental point is to have them smaller than 8 [$Bytes$] for a reason that will be explained later. As a remark, these messages do not follow the encoding that will be presented below.

4.2 Data Encoding

One of the critical points is to reduce the number of bytes used by the data buses to transmit information. The initial idea was to simply encode all the data in a specific format by employing ASCII characters (Assuming 1 [$Byte$] per character). It is a convenient way to encode and decode the message and allows an easier implementation. In addition, it decreases the complexity of debugging. The primary drawback is, of course, the number of bytes needed to encode a straightforward message. One of the primary weaknesses is linked to the limitation of some data buses to a maximum of 8 $Bytes$. For example, if the PF OBC needs to send a UNIX timestamp, it is impossible to do it in one packet with ASCII characters. Indeed, the timestamp contains 10 digits which would mean 10 bytes.

In this study, the idea has been to follow a similar encoding method as Binary Coded Decimal (BCD) arithmetic[164]. It furnishes an elegant and efficient way to code decimal numbers into binary. The basics of encoding numbers directly in their binary form and not having a fixed number of digits to be sent are kept. In addition, the requirement to reserve some character

for another purpose like data separation or end of packets is implemented. For these reasons, the numbers are first artificially split into sets of two digits and then converted separately into binary. It is critical to notice the data are consistently in pairs which means potential inefficiency for an odd number of digits. As example, the integer 12345 is split into 01|23|45 and then converted to $0x01|0x17|0x2D$. It means uniquely 3 $[Bytes]$ are required instead of 5 $[Bytes]$. One can argue that converting the whole number into binary is advantageous (in this case $0x3039$), but it would imply using integers higher than 99 or $0x63$. Indeed, it could result in sending a number such as $0x3aa$. By consequence, employing a dedicated character for packets' separator would be difficult to distinguish from part of a number. This particularity is addressed later. Ultimately, a unique case is done regarding the number 0 since $0x00$ means end of strings in ASCII. To prevent this effect, the number 0 is encoded as 100 or $0x64$.

In addition to encoding positive numbers, the special character $0xEE$ is used at the beginning of the data to indicate a negative value. Regarding floating-point values, the encoding is a bit more difficult. The first task is to split the integer part from the decimal one. The integer part is encoded as previously explained. Then, the special character $0xDD$ is used to indicate the separation. Ultimately, the decimal part is equally converted to binary in pair of digits. For example, the number -123.456 is taken. Initially, it is split in pair $-|01|23|.|45|60$. It is decisive to identify the specificity in the decimal part as the number's splitting differs slightly. In this case, the extra 0 is added at the end if needed since $0.4560 \neq 0.0456$. The following step is as before to convert the number into binary $0xEE|0x01|0x17|0xDD|0x2D|0x3C$. In this case, the encoding has the worst ratio possible since both the integer and the decimal part contain an odd number of digits.

As mentioned earlier, some particular values are used for dedicated functions. The list below presents them.

- $0xAA$ is defined as padding. If a packet needs some extra bytes without any information, this value is used.
- $0xBB$ is the delimiter between packets. If a request has more than one, the software sets this character. It allows to more easily split the list of packets.
- $0xCC$ means the end of the request or reply. It is the last value of the last packet transmitted. It allows the receiver to clearly recognize no other packet will be sent.
- $0xDD$ intends as the replacement for the 'dot' in a floating number.
- $0xEE$ is the indication of a negative number.
- $0xFF$ is the information delimiter between values in a packet. This delimiter is used to separate each meaningful data in a packet. (For example, two different numbers)

The ultimate point to mention is the splitting of a message into multiple packets. Since all the data buses have not the same data length, it is crucial to have the ability for the encoding to be adapted. It typically means that both the data request and reply are split into multiple packets.

Finally, arguments can be made against developing an entirely new encoding method. Indeed, it requires time and effort to develop and test. Nevertheless, it was estimated to be straightforward to implement and allows better flexibility regarding the data sent.

4.3 Data Request

The encoding of data is split into two fundamental categories with similar implementation. The first one is the data request by the PF OBC as presented in the previous section. Its role is to efficiently demand information from the PL OBC. The basic is to request the information regarding a certain number of algorithms inside the PL OBC. It assumes that each of them produces a single computed value which is stored inside the data pool of this computer. The principle is that the PF OBC request the measurements of the algorithms by sending the ids linked to them.

First, all packets containing the request or part of it start with the letter $R\ or\ 0x52$. At that point, there is the indication of the packets' number or the current id. It has been decided that the first request contains the total number of packets sent. Then, the following ones contain their id in the second byte position. It is important to notice that the maximum number of packets that can be sent in one request is 100 since the id is coded in two digits binary following the method described before. Right after the two initial bytes, the timestamp of the request is encoded. At this point no packet delimiter is used since the beginning is consistently on two bytes. After this, the number of requests is included, and then the list of ids as explained in the previous section with every time the information delimiter. Ultimately, a single end of the message is added. The Figure 2.7 shows a typical encoding for a 3 packet requests of 3 ids ($0x03$) with a maximum of 8 [$Bytes$]. In this case, the ids are 2 ($0x02$), 4 ($0x04$) and 5 ($0x05$). The limitation in the number of bytes is directly linked to the specific data bus used. In this example, the limit of 8 [$Bytes$] imposes to split the request into 3 packets.

4.4 Data Reply

The data provided by the PF OBC contains computed values or measurements of algorithms. It is assumed that each algorithm produced a single type of data. In addition to this data, information is added regarding the timing and the origin of the value.

The reply of the PL OBC is equally structured in an identical way. In this case, all packets start with the letter $M\ or\ 0x4D$. (The 'M' is for measurement.) As for the request, it is directly followed by the total number of packets sent in binary form. If it is not the first one, the current id is in the second byte. At that point, the timestamp of the answer is added without any delimiter either. Finally, there is a delimiter and the number of measurements that are

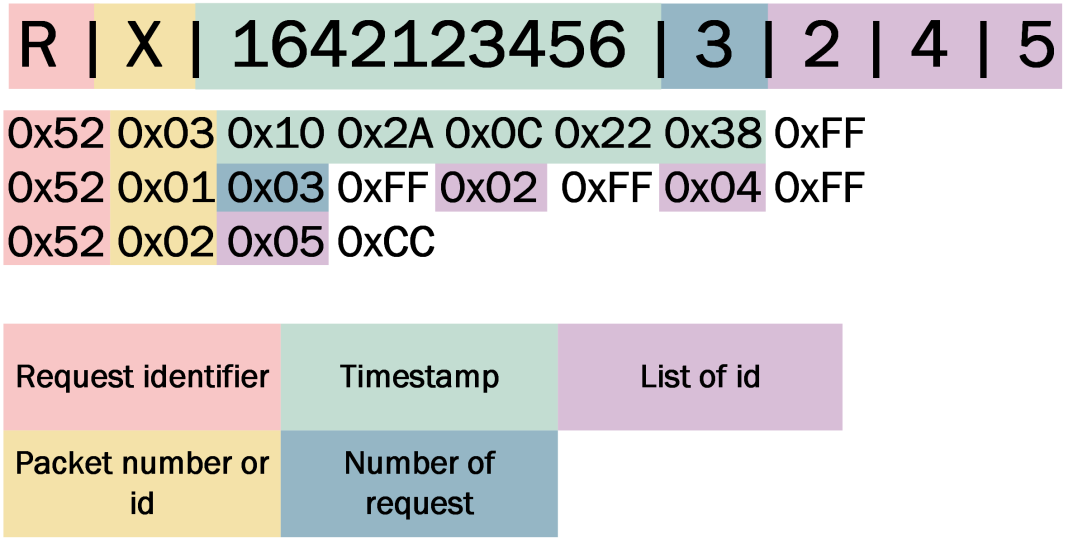


Figure 2.7 – **Encoding of the data request.**

contained in the reply message. This initial part represents the core of the message.

After that, each measurement is encoded in the message in the requested order. For one of them, there is initially the id of the algorithm, the id of the value, and the value itself. It is followed by the timestamp of the computed value and the timestamp of the measurement. Between each element, the $0xFF$ delimiter is used. In the last packet, the end of the message is included. A typical packet example can be seen in Figure 2.8 with the reply of one measurement with a maximum of 8 $[Bytes]$. Regarding the measurement data (purple), the first element is the id of the algorithm (2 or $0x02$), then there is the id of the value (12 or $0x0C$). The next element is the value (-13.24 or $0xEE$ $0x0D$ $0xDD$ $0x18$) and the two timestamps on the following lines.

5 Data Buses Implementation

This section describes, in brief, the various data buses and their custom implementation in both the PF OBC and the PL OBC. More information about the data buses and their specifications/standards is provided in the state of the art.

As a first notice, the SpaceWire network was not tested in this chapter for two reasons. The first one is that it was already presented in the previous chapter of the thesis. Even though no extended analysis has been conducted, the data bus was tested on many occasions. In addition, Airbus DS GmbH, Friedrichshafen is currently working on a similar HIL testbench that has SpaceWire as the main link. In the end, it was estimated to be redundant work to analyze. The second reason is regarding the differences in complexity from the other ones. The goal of this chapter is to look at a fall-back communication channel and it would not be representative

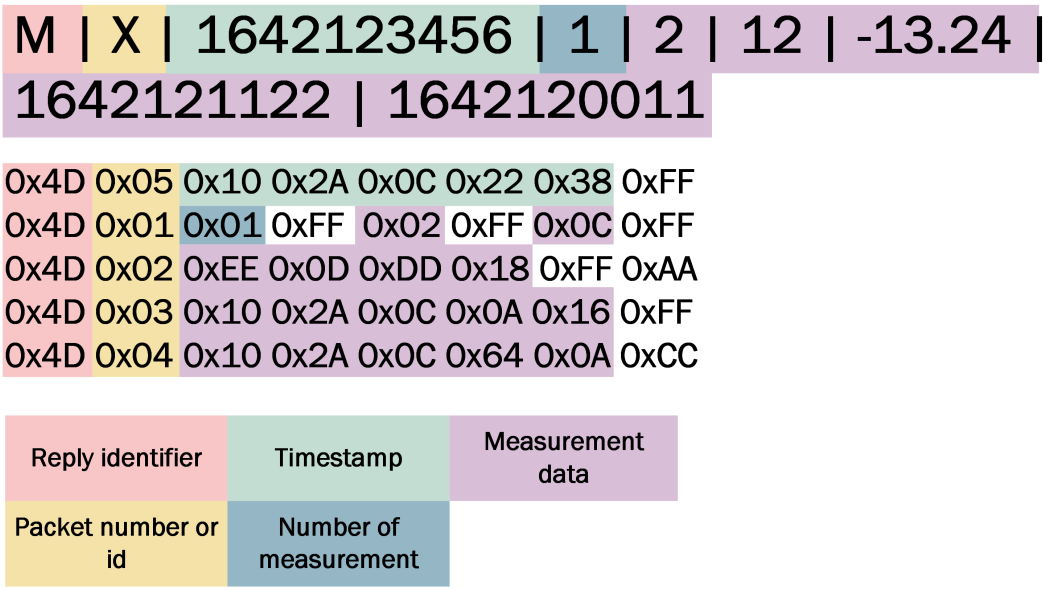


Figure 2.8 – **Encoding of the data reply.**

to include the RMAP/SpaceWire protocol. For the same reason, the CCSDS Packet Transfer Protocol is not investigated. Nevertheless, the comparison would be interesting to understand the limitation of these fall-back data buses options. As a reminder, a SpaceWire network can support a bit rate between 2.0 and 200 $[Mbit/s]$[165]. In addition, the Frame data has no limitation in the number of bytes[165].

Regarding the data buses tested below, the goal is to first establish a few requirements. Since their target is mainly a fall-back from the main communication channel or alternative for cheaper ADR mission, the data rate is less crucial. In an ideal situation, the PF OBC should receive measurement every second to update its GNC algorithm, especially when relative navigation is desired. Nevertheless, if the fall-back communication channel has to be used, then the operators of the spacecraft will probably be extra cautious. It means all the maneuvers might be slower, and every spacecraft decision is meticulously planned. By consequences, the rate of measurement can be slower and the arbitrary target of 1 measurement every 3 seconds is chosen. Nevertheless, it is fundamental for this fall-back system to provide accurate information to the Platform On-Board Computer. It means all the measurements sent should have a high enough precision. Finally, the last requirement is that 99% of the data received should be valid. At this point, no constraint is fixed regarding the number of failed messages, even though it is obviously a significant point. It is estimated more important to receive the correct information than no information at all.

For all the data buses presented below, some dedicated libraries are used for their implementation. The goal is to cover the gap between the OSI model layer 2 provided by their respective standard and the application layer (No. 7) presented in the "High-Level Data Exchanged" subsection.

In all the implementations described below, a limitation on the number of bytes per packet is imposed. This constraint directly comes from either the data bus standard or the library used. Nevertheless, the high-level protocol as well as the encoding support any number of data bytes, and thus no specific modification is required.

5.1 I2C

The I2C data bus is admirably suited for a variety of applications. With its two lines, it can even support multiple masters - multiple slaves architectures.

For the development of this testbench, the PF OBC is using the "i2cmst" library provided by Cobham Gaisler AB. The usage and implementation of the data bus are reasonably straightforward in an RTEMS architecture. The first step is to register a driver in the software and then it can be open in a standard way. There are no specific parameters to set, and the communication can start straight away. For this data bus, the limitation regarding the number of bytes per message is set to 16, even though the library decomposes it into smaller packets. Regarding the usage of the library, the standard functions $read(..)$ & $write(...)$ are used to send and receive packets. No specific constraints are needed in this case to employ it.

For the PL OBC, the "PIGPIO" Library[166] is operated to ease the implementation. It provides a low-level interface to the slave data bus implementation and is straightforward to implement. It implies the use of a read and write function combined. In this function, the transmitting buffer is given and its content is sent if data are being received. To start the bus, a simple call for library initialization is needed. At this point, the configuration of the port with the correct parameters for the slave's address is done.

Because of the bus' instability, some modifications have been implemented in the data exchange protocol presented earlier. Indeed, the bus often skips or misses a packet when the exchange is ongoing and it is causing multiple issues. To prevent most of the failed transactions, the master is directly requesting the ids of the packets that need to be received. This extra step is particularly required when the reply message is sent to the platform processor. The effects of this implementation are described in the following section.

5.2 SPI

The SPI bus is a well-known standard used in embedded electrics. It is frequent to see a processor board connected to multiple external devices implementing its master-slave architecture. It requires four lines with the inclusion of the chip select.

In the PF OBC, the "spictrl" library developed by Cobham Gaisler AB is used. It allows an easy implementation of the data bus. The first step is to register the driver for the SPI application and then open it in the read and write modes. For this application, the baud rate is set to $200'000$ $[bits/s]$, and the limit of 12 $[Bytes]$ per message is used. As for most of the data buses

described here, the basic "C" function $read(..)$ & $write(...)$ can be used to communicate with the port.

In the PL OBC, the "PIGPIO" Library is again used. The functions are actually the same as for the I2C implementation, and it uses the low-level interface for the slave peripheral. To open the port, the library is initialized and then configured with the polarity and phase parameter of the bus. This data bus has required the same modifications as in the I2C case because of its instability. The root of the instabilities might be linked to the library used or might directly come from the payload electronic board.

5.3 CAN

The CAN bus is a data bus widely used in the automotive industry. It is possible to connect a wide range of devices and its structure allows a very defined interaction between the various components.

For this application, the PF OBC is using the OpenCores CAN driver interface implemented by Cobham Gaisler AB. It makes the utilization of the CAN bus easier and prevents redefining some parts of the interface. In addition, since the bus contains two nodes, some parts of the data bus can be simplified. The first step is to open the CAN port and then set up the transmission speed to $250'000$ $[bits/s]$. The following one is to put the data bus in non-blocking and remove all filters. One of the particularities of the CAN bus remains its ability to set filters on addresses to receive particular messages. Since this setup contains two elements, these functionalities are not needed.

For the data bus usage, the read and write function of the OC CAN Library are used in the PF OBC. Regarding the packet themselves, it has been decided to work with the base frame format instead of the extended. This decision is made since the interaction is purely between two components. In addition, the CAN frame is sent on a single instance and accept up to 8 $[Bytes]$ of data.

In the PL OBC, the basic CAN Library from Linux is used. Despite using an external module to enable this communication, the core software does not require any modification. The bit-rate is set to the same level as the PF OBC and then the port is open in "raw" mode. As for the other board, no filter is implemented for the sake of simplicity. The primary advantage of this library remains the ability to utilize the standard "C" function for read and write operations.

5.4 RS-422

As described earlier, this data bus is ordinarily used for device connection since it is a single master - multiple slaves. Moreover, it is widely operated thanks to its straightforward implementation and wide availability on the market.

On the PF OBC side, the primary software needs to open a link to one of the default drivers of the board. The following step is to retrieve the current parameters of the data bus. Among the fundamental variables to initialize is the size of the bytes which should be 8 [$bits$]. In this implementation, most of the special modes and functionalities are disabled. The maximal blocking time for the read function is set to 1 [s] and the baud rate to 9600 [bd]. Once all the parameters have been configured, the port is ready and the communication can start.

Regarding the send and receive packets, the native "C" function $read(..)$ & $write(...)$ are again used to access the port. To be noticed that the limit of 8 [$Bytes$] per message is used in this application. The last implementation detail worth mentioning is the read function is getting data byte per byte to ensure receiving the entire packet. Indeed, some issues were happening with incomplete packets when the data bus tried to receive the full message at once.

For the PL OBC, the setting up of the data bus is the same. The noticeable difference is the use of the extra module to enable it. Nevertheless, the driver implementation is essentially following an identical procedure and setting up matching parameters. As for the main board, the size limit is set to 8 [$Bytes$]. For the data exchange, the native "C" functions are again used to send and receive packets on the port.

6 Data Transmission Efficiency

6.1 Procedure & Data Buses

To assess the efficiency of the data buses, a procedure has been developed. The goal was to monitor the data exchange between the PF OBC and the PL OBC during a number of transactions. The monitoring was done regarding various timings such as the time to send a request or to prepare a reply. In addition, the control of the data received by the PF OBC was done after the test phase. It guarantees the correctness of the information and ensures no mistakes have happened. To do so, all the reply messages received by the PF OBC are stored in the development workbench and then compare to the generated data on the PL OBC.

To obtain a certain level of reliability, the procedure asks for 100 transactions between the PF OBC and the PL OBC. The platform processor allows a few seconds after the end of each transaction to eliminate some non-predictable effects. On the other side, the payload computer is simply waiting for a request and reply as soon as the correct message is obtained. At this point, it is crucial to save the received request as well as if it has been decoded successfully. An essential element to mention is that all timings have been measured with the internal clock of the PF OBC/PL OBC. This fact means the measurements have a precision of 1 [s] and can not be considered highly accurate. Nevertheless, the timing has been saved to enable comparisons of the various data buses. The general results of the four data buses are presented in the following subsection.

The first intriguing part to examine is the number of packets exchanged for each type of data

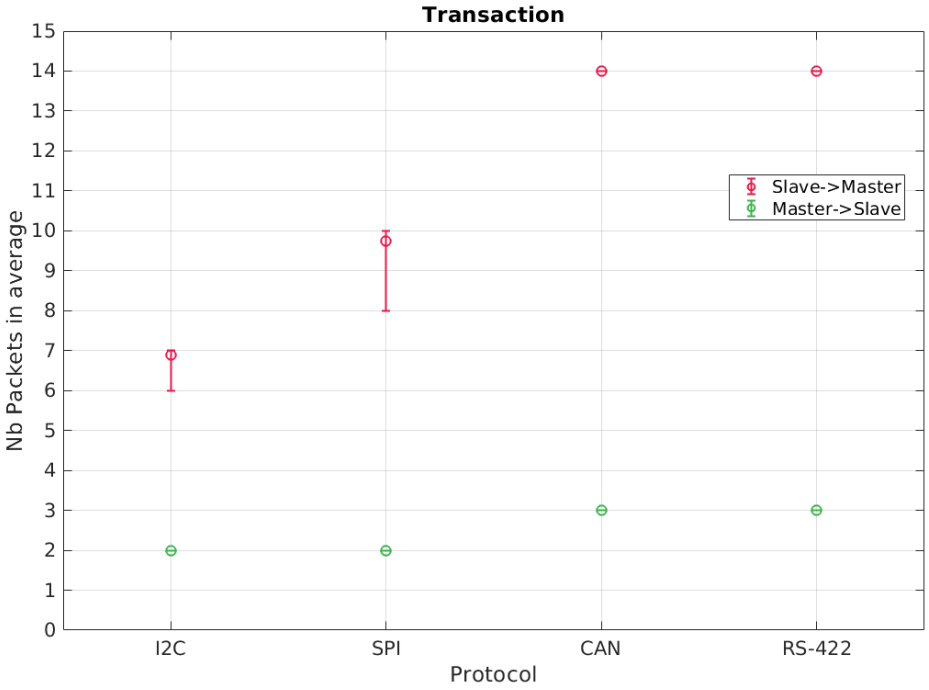


Figure 2.9 – **Data transaction of all data buses (in average).**

bus. Figure 2.9 shows a summary of the average data packets transmitted. An essential point to mention is that failures in the transaction are not shown here. The value indicates the number of different packets that have been created by the master or the slave to send the information. The variation between the data buses is directly linked to the number of bytes per message allowed in the software. The CAN and RS-422 data buses are uniquely allowed to use data packets of 8 [$Bytes$] instead of the 16 [$bytes$] for the I2C (or 12 [$bytes$] for the SPI). It means they require on average about twice the number of packets. The variation of the slave to master in the I2C and SPI data buses is linked to the size of the data. As mentioned before, float values are encoded with a maximum of 6 [$Bytes$], however, they can require less space. It means depending on the values sent, the number of bytes used varies and can affect the number of packets created.

6.2 I2C

The first data bus to be looked at is the I2C. Figure 2.10 displays the timing of various phases of one transaction and the validity of the exchange from the PL OBC perspective. Regarding the timing, the time to receive packets is between the start of a test and the reception of all messages. To be mentioned that a test starts every few seconds and finishes after a brief moment if no data have been received. For this reason, the duration of the first phase varies from 1 to 3 seconds. It is directly dependent on the speed at which the main PF OBC requests

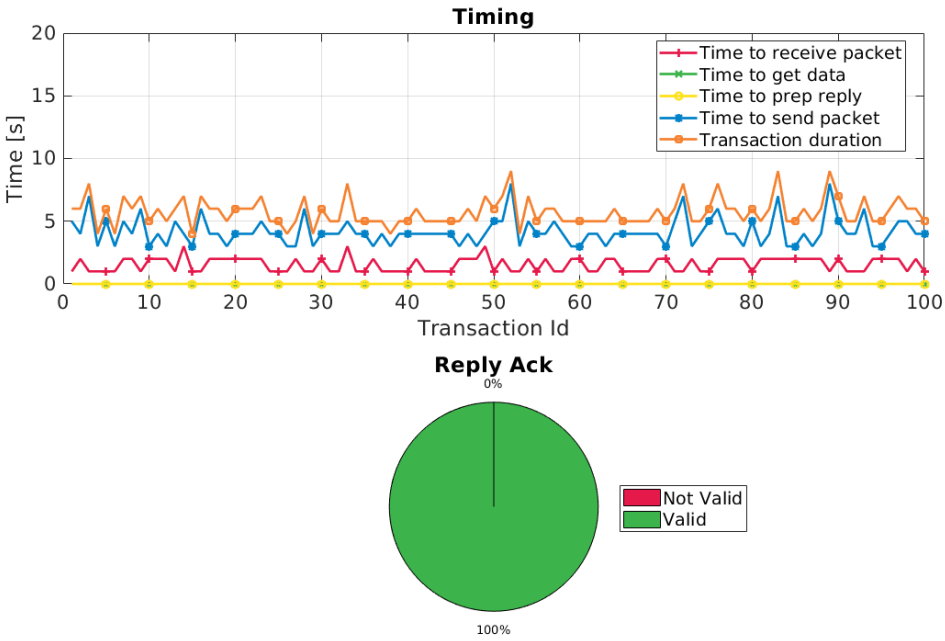


Figure 2.10 – **I2C - PL OBC timing and validity.**

data. An excellent point is that both the time to get the data and the time to prepare the reply are consistently zero. This timing is consistent in the PL OBC for all the data buses tested. The meaning is that the request decoding and the reply encoding are both efficient processes. As a conservative estimate, their timing can be neglected in comparison to the other actions of the payload computer. The next timing represents the time to send packets that shows the duration needed by the PL OBC to transmit all the data to the main PF OBC. This measurement is performed between the ends of the data encoding until the master acknowledges all the packets. It has been seen that sometimes the acknowledge message is received late and therefore the test duration is increased. Nevertheless, the significant variation in the timing is linked to the unstable transaction of this I2C implementation. As it has been described earlier, the data bus suffers from some issues with failed transactions. Consequently, increased complexity has been added to ensure a higher level of correct transmissions. It would mean resending the same packets if the PF OBC identified some error. The second part of the plot display the success in transmitting all the messages. A valid result is registered if the master sent the acknowledged packets. It does not include errors in the reply itself. It means the PL OBC has received the confirmation of the received message from the PF OBC. In this case, the 100 transactions have been successfully accepted by the PF OBC.

Figure 2.11 shows the various timings for the PF OBC with the I2C data bus. The first timing listed stays consistent for the whole test at 1 $[s]$. The creation of the request does not represent a complex action and thus the duration does not fluctuate. The second one is the time to send

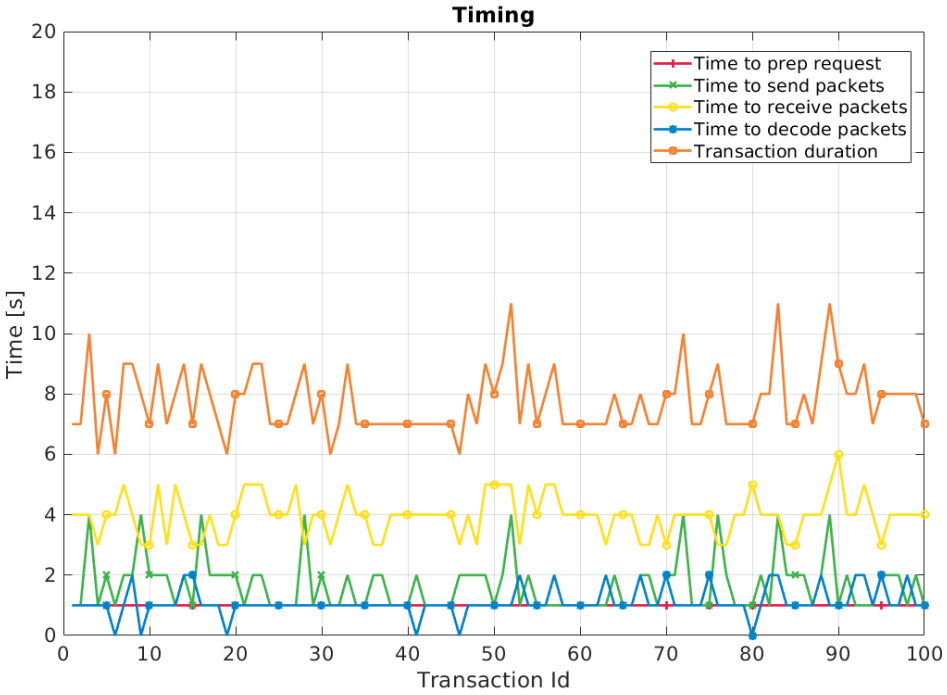


Figure 2.11 – **I2C - PF OBC timing.**

the packets. In this case, the variation is between 1 [s] and 4 [s]. This effect is again linked to the unstable implementation of the I2C data bus in the setup. The PF OBC has to send occasionally multiple times the same packet before it is correctly received by the PL OBC. In the same category, the duration to receive the reply has some fluctuation of 3 [s]. The reason is similar to the previous timing but it is now the payload that has to resend multiple times the same packet. Ultimately, the last timing is surprisingly varying during the test. Despite the decoding of the message being a standard process, it seems the PF OBC needs sometimes up to 2 [s]. It might be linked to the validity of the data structure and the fact that sometimes uniquely part of the reply is received. Overall, assuming that the general transmission time is about 8 [s] and that every message carries 3 measurement, the timing requirement is not met. Nevertheless, since no special care was placed on the timing design, it is a promising result.

The last intriguing point to examine is the validity of each transaction for the I2C data bus. Figure 2.12 has the three different pie plots to present the results. The first one represents if the PF OBC has acknowledged all the transactions. It means the master had received all the expected packets and sent the confirmation to the PL OBC. In this case, the result shows that 100% of transactions have been correctly received. The second plot reports the percentage of interaction where the data structure is valid. For the I2C data bus, up to 8% of the transaction had a validity issue. It can be erroneous in various forms. The structure of the reply could have missing elements or some delimiter might be lacking. If the message has not a valid structure, the information is discarded by the PF OBC. Once all messages have been checked, the PF

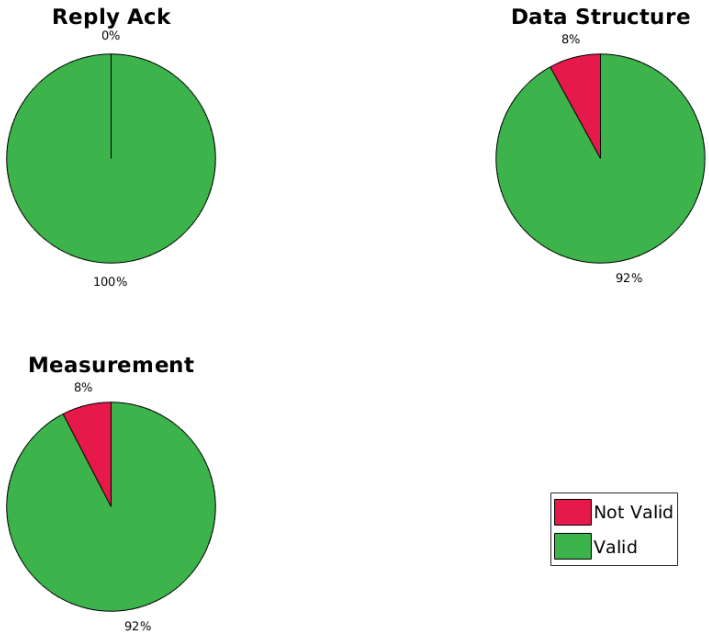


Figure 2.12 – **I2C - PF OBC validity.**

OBC extracts the data from the valid packets. At this point, a measurement can be erroneous if one of the values differs from the one generated by the PL OBC. Another option is if some part of the requested data is missing. For example, the reply message uniquely contains two valid measurements instead of the three expected. A significant point to recall is the validity check is done in this test and can not be implemented in a concrete situation. The PF OBC controls that all the requested information has arrived but can not verify if the values are exactly the ones generated by the PL OBC. As explained earlier, a procedure exists for the PF OBC to check the measurements. Nevertheless, if the value is in an expected ranged, no error is produced. For the I2C data bus, about half of the erroneous measurement comes from errors in the main value. The rest are from missing measurements. Overall, it means the 99% requirement regarding the data validity is not met. Looking at the measurement control, the I2C gets a 92% of its measurement valid. At this point, the structure validity is not considered since the PF OBC can verify its integrity and discard it if needed. Nevertheless, it is an issue that needs to be worked on if this data bus is considered.

6.3 SPI

Figure 2.13 shows the behavior of the PL OBC with the SPI data bus. As for the I2C, the duration to get the data and the time to prepare the reply are both consistently at zero. It is a predicted result since both processes do not depend on the data bus selected. The time to receive

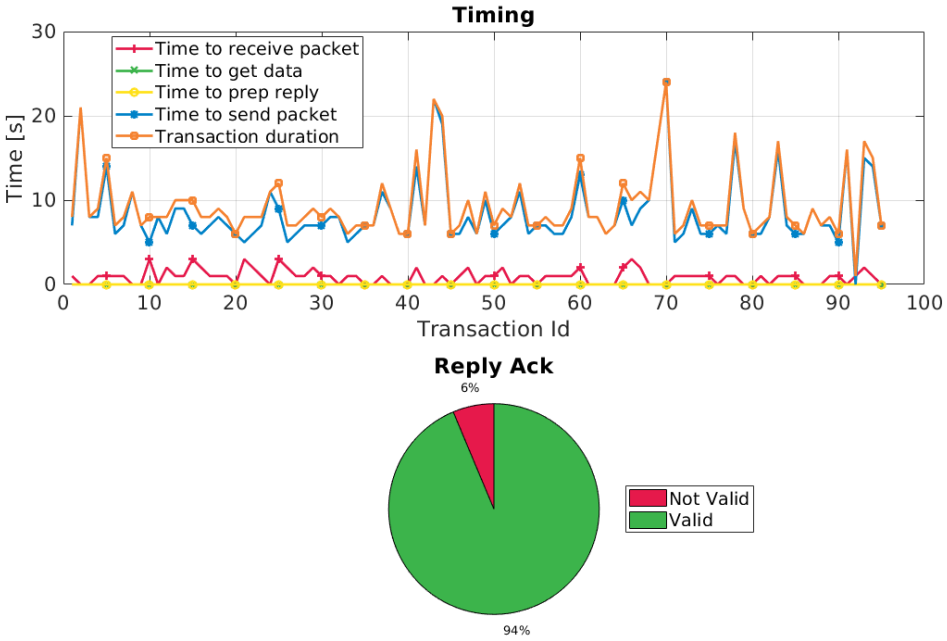


Figure 2.13 – **SPI - PL OBC timing and validity.**

the request undergoes some slight variations. They are linked to the overall instability of communication. Similar behaviors can be observed for the time to send the packet. By further investigating the data exchange, the issue reveals to be linked to some errors in the message received on both sides. It happens on multiple occasions that a completely wrong packet is received and thus the transaction has to be restarted partially. In addition, some packets are lost in the exchange implying additional steps for the PF OBC to request it. To be noticed that the figure displays less than 100 cases since some transactions never happen because of failures. The reply acknowledge pie chart reflects the general issue of the data bus. In this case, 6% of the data exchange is not acknowledged by the PF OBC to the PL OBC.

The various timings of the PF OBC with the SPI data bus are presented in Figure 2.14. Similar to the payload computer, there are many variations between the transactions. In comparison, some of the timings are more or less consistent throughout the whole test. The time to prepare the request and the duration of the decoding are not related to the data bus implementation and thus have slight variations. The duration to send packets undergoes some fluctuations even though its amplitude is attenuated by the scale of the plot. The most remarkable behavior comes from the time to receive packets. The variations are up to a factor 8 in some cases. In addition to the reasons mentioned before, the master struggles to receive the stop packet in some cases. With a deeper analysis of the interaction, it seems the PF OBC can be stuck in a loop waiting for the last message of the payload. In most cases, the packet is simply lost. This lost message implies additional time on the PF OBC to receive the correct packet

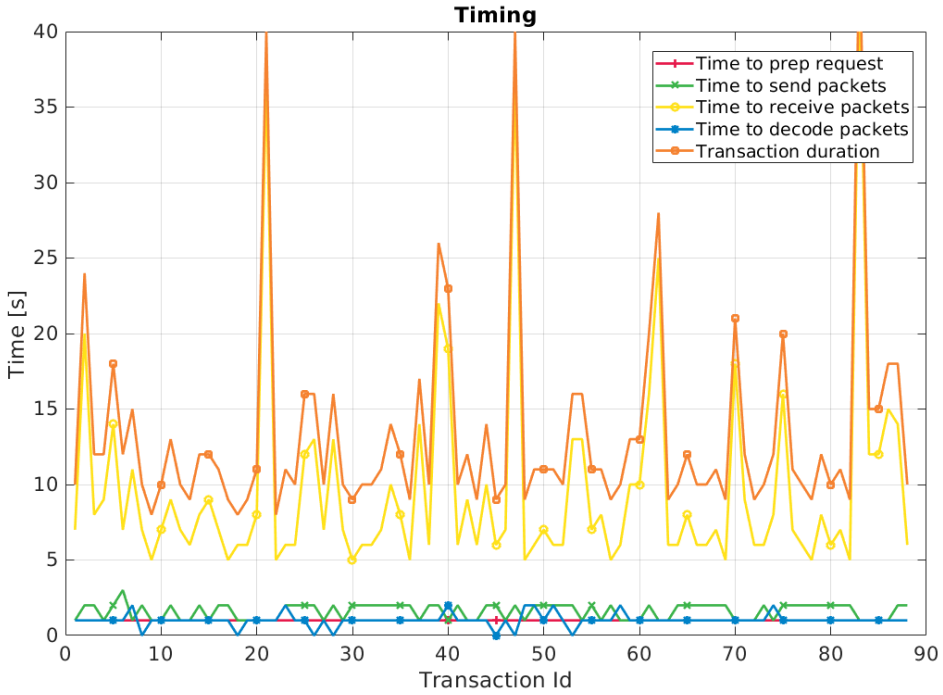


Figure 2.14 – **SPI - PF OBC timing.**

and therefore creating these spikes in the figure. The reason for the lost packet is currently unknown, nevertheless, suspicion is on the sending hardware of the PL OBC. By analyzing the signal on the lines with an oscilloscope, the instrument shows that some packets seem to be never generated by the PL OBC. More investigation should be conducted on the issue. As for the I2C, some additional processes are implemented in the transaction to reduce these effects, but a few issues remain. With the timing result even worst than for the previous data bus, the corresponding requirement is not met by almost a factor of two.

The last results regarding the SPI data bus are in Figure 2.15. One surprising element remains the difference in acknowledgment between the PL OBC and the PF OBC. The platform has 2% more transactions without a valid ack packet. This issue might come from the master not receiving properly the $stop$ package despite forwarding the end message. In addition to this unusual behavior, the master has received 15% of invalid data structure. It is one of the highest of all data buses and is directly linked with the instability of the implementation. Nevertheless, the validity of the measurement is at 100%. This significant result is correlated to the nature of the failure happening in the data bus. Some packets are wrong and incorrect but there is never one or two erroneous bytes that would corrupt the measurement. In all cases, the errors modified the whole reply to the point where the data structure is not valid anymore. Technically, this data bus met the data validity requirement. As mentioned earlier, the requirement is not looking at the reply acknowledge or the data structure since both can be detected by the PF OBC. For this data bus, all the measurements received were valid.

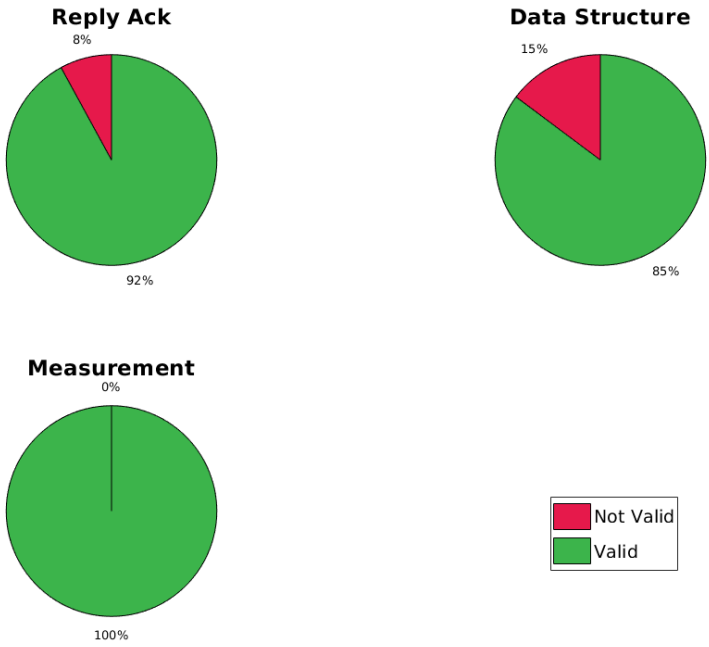


Figure 2.15 – **SPI - PF OBC validity.**

6.4 CAN

The CAN data bus implementation in the payload shows some promising results in Figure 2.16. As for the other ones, the time to prepare the reply and to get the data are consistently zero. The change of data bus is not influencing this part even though more packets are to be created or decoded. The reception duration is around $2-3$ $[s]$ and can suffer from a similar issue as previously mentioned. Meaning a test could have started sometime before the PF OBC initiates the transaction. Regarding the sending duration, the results indicate a constant 5 or 6 seconds delay. To be noted that about twice the number of packets have to be sent in comparison to the I2C. In any case, the timings are extremely consistent in this implementation. Regarding the validity, all the transactions have been accepted by the master with the acknowledge packet.

For the implementation in the PF OBC, the timings are presented in Figure 2.17. Two of the timing are at 1 $[s]$ for the entire duration of the test. The time to prepare the request is consistent and does not vary. This standard process does not depend on the transaction. The time to send packets is a bit more surprising. It could be expected that the request part of the test includes an unknown duration depending on the state of the two computers. Nevertheless, it is not the case for the CAN data bus. Regarding the time to decode, the duration is fluctuating around the 1 $[s]$ mark. The result is expected since it is a standard process. It is also similar to the I2C data bus since the number of packets does not influence the length of the message. The last crucial timing to consider remains the time to receive the packets. It is again surprisingly

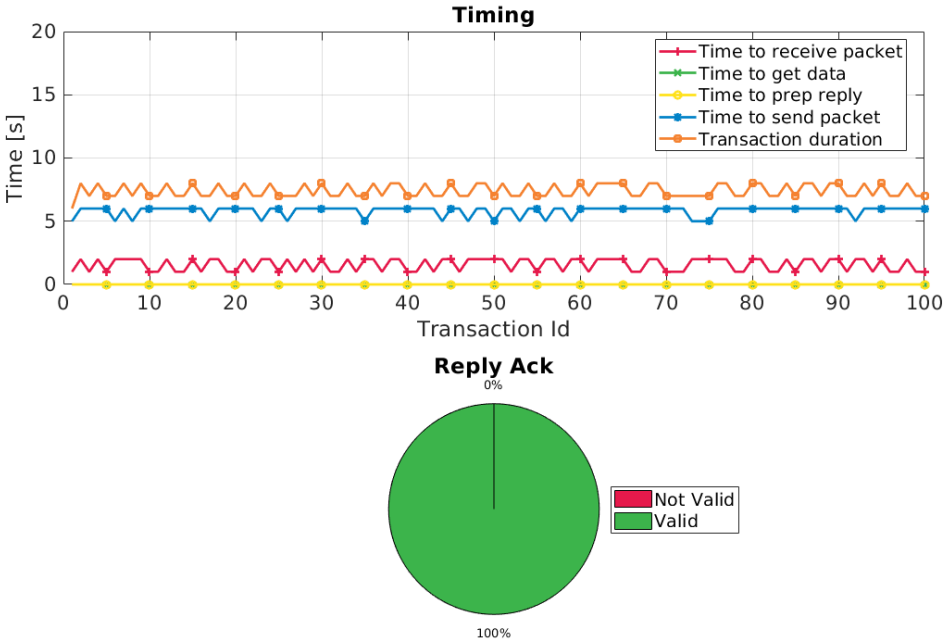


Figure 2.16 – **CAN - PL OBC timing and validity.**

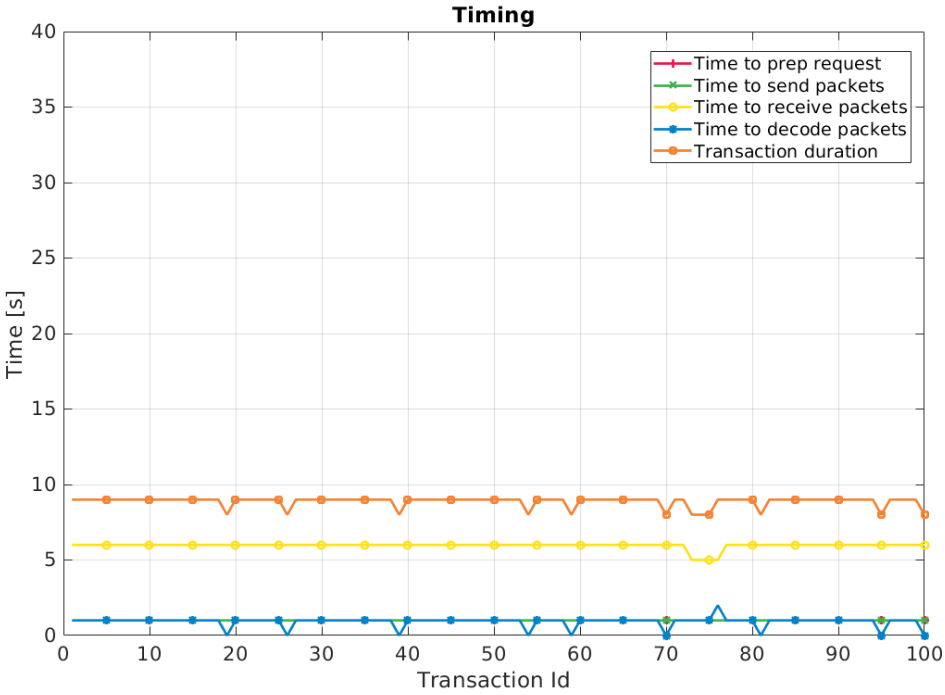


Figure 2.17 – **CAN - PF OBC timing.**

consistent among all the transactions with a steady value of $5-6$ [s]. The lack of fluctuation in the timing represents a promising aspect regarding the CAN data bus. Regarding the timing requirement, it is also a promising result. Since three measurements are sent per transaction, it means the CAN data bus is compliant with a maximum of 9 [s] for one transaction.

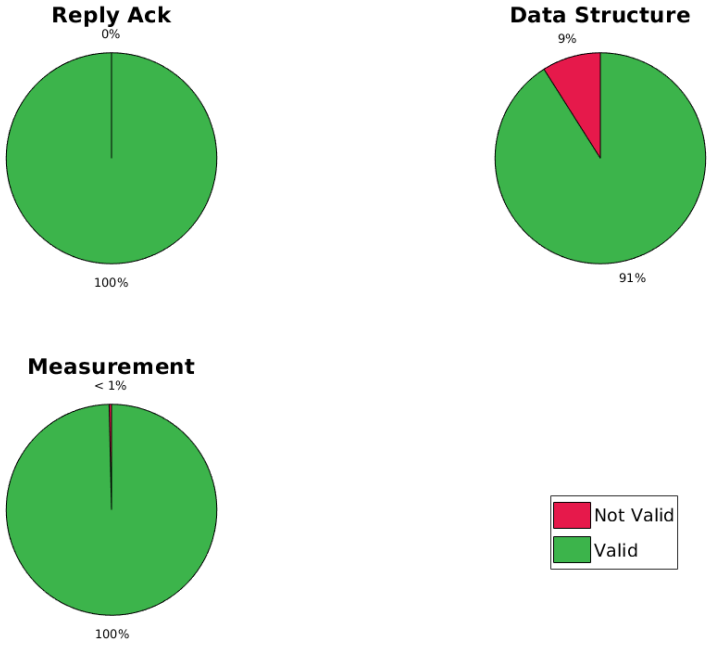


Figure 2.18 – **CAN - PF OBC validity.**

In the validity domain, the implementation of the CAN is encouraging. Figure 2.18 contains the result of the hundred transactions. The first pie chart displays a predicted result with all the transactions acknowledged by the master. Regarding the data structure validity, there are slightly more erroneous messages than for the I2C one. A possibility could be the increasing number of packets needed for each transaction and therefore a higher chance of one being not correct. A relation can equally be perceived between the decode timing and the number of failed packets. For this data bus, exactly 9 transactions had a time smaller than 1 [s] for the decode phase and there are also 9 invalid structures. Despite this result, the CAN data bus is demonstrating some satisfactory performances. Finally, the statistics of the measurements show less than one percent error with uniquely one erroneous data. The root of the error might come from a corrupted byte in the data. This result is also very promising regarding the validity requirement. It shows that the CAN data bus can almost reach this level. Nevertheless, some additional work needs to be done regarding the data structure validity.

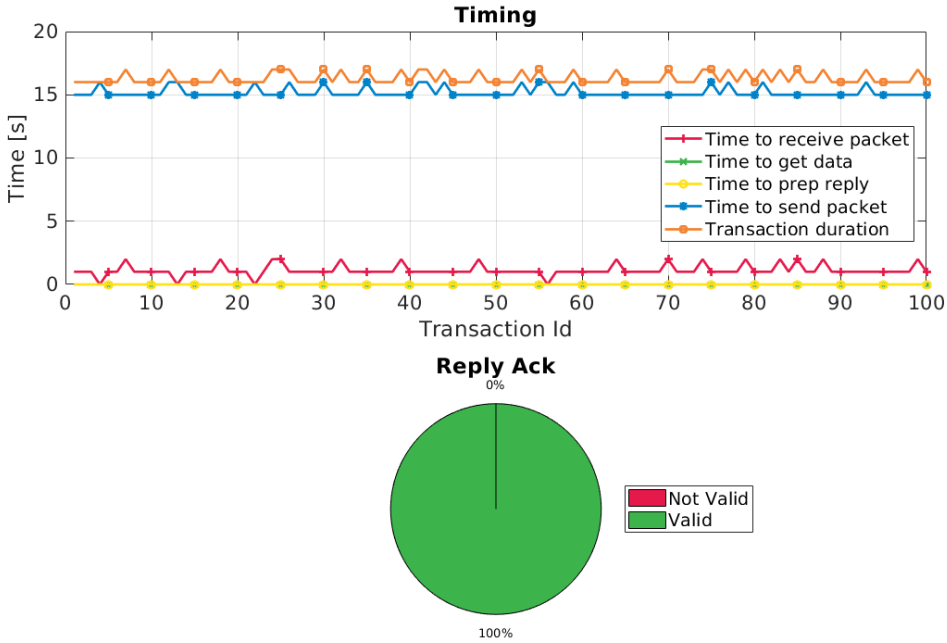


Figure 2.19 – **RS-422 - PL OBC timing and validity.**

6.5 RS-422

The last data bus to be inspected is the RS-422 with first its payload implementation. Figure 2.19 shows the timing and validity of the whole test. The get data and prepare reply have again a neglectable duration in the test. Both plots stay at 0 [*s*] for all the transactions. The time to receive packets is also quite low with an average of 1 [*s*] with sometimes even below this number. The highest of all timings remains the time to send the packets. This duration varies between 15 and 16 seconds. This particular effect might be biased because of the implementation of the data bus. Indeed, some instability would happen in the transaction if the packets were transmitted with a high frequency. A solution identified has been to require an arbitrary pause between each packet transfer to reduce the risk. This design choice effect is shown here. All things considered, the consistency between each transaction is high, but the duration is not ideal. Regarding the acknowledgment of the master, all the transactions have been accepted.

Looking at figure 2.20, all the timings for the PF OBC undergo low fluctuation. The time to prepare the requests and to send packets are both at 1 [*s*]. The first one is expected regarding the result of the previous data buses. The second one is similar to the CAN with a surprisingly fast request transmission. In addition, the time to decode a packet has a low duration between 0 and 1 [*s*]. Since it is a standard process, the result is expected. Ultimately, the time to receive packets has a steady 15 [*s*] duration during the whole test. The consistency of this timing is

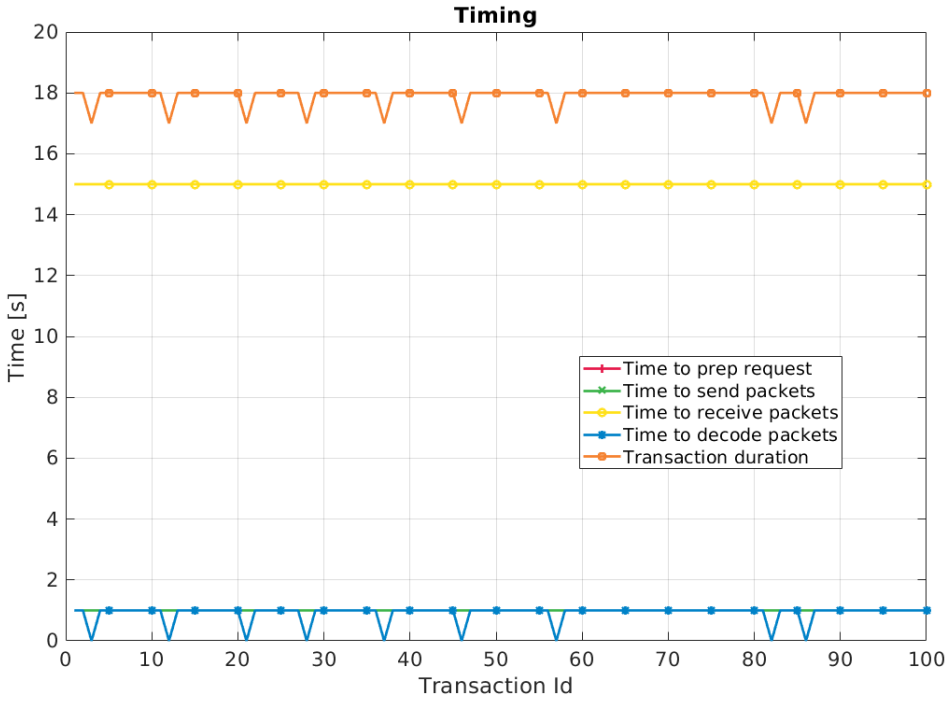


Figure 2.20 – **RS-422 - PF OBC timing.**

surprising and shows that almost all packets in a single transaction are received on their first try. As described before, the superior overall values are linked to the data bus implementation. With this timing, the requirement is not met by a factor of two. Extensive analysis should be done if this bus needs to be chosen.

The Figure 2.21 shows some promising results regarding the validity of the transactions. The reply acknowledge pie is as usual at 100% meaning all the messages have been received correctly. Nevertheless, the error in the data structure is at 15% which is the highest of all data buses. In this implementation, the structure of the overall reply message is corrupted more often than with the others. The surprising result is that the measurements inside each valid reply are always correct. The data bus scores 100% validity for the data encoded inside the message. Technically, this data bus met the requirement for data validity. Nevertheless, it seems that the critical issue is linked to the general structure of the message itself and not its content.

6.6 Comments

On the general aspect of all data buses, it seems the consistency of the timings can be an excellent indicator of the actual bus efficiency. The I2C is the one with the extremest fluctuation and has the highest measurement decoding issue. Nevertheless, the overall timing is the fastest.

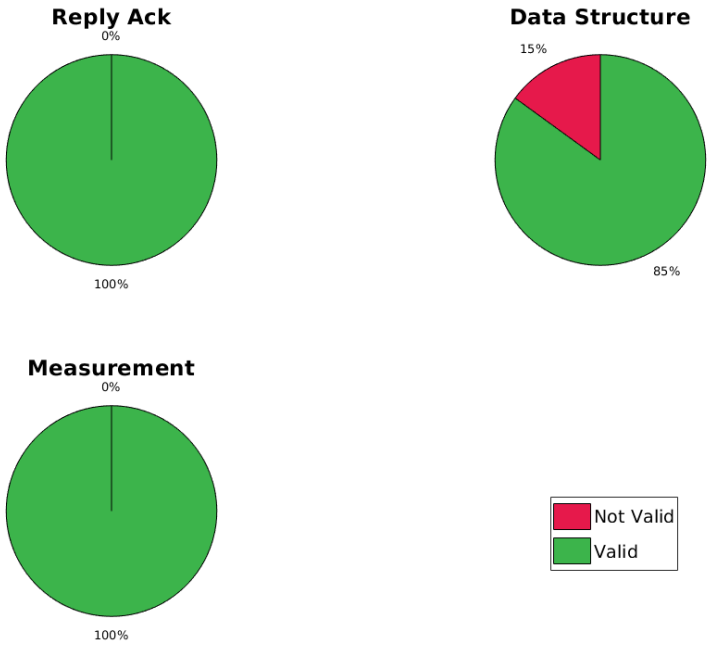


Figure 2.21 – **RS-422 - PF OBC validity.**

A fundamental point to recall is that the number of packets needed to transmit in this data bus is lower. Indeed, the data limit of the I2C is twice as big as for the CAN and RS-422 data buses. Identically, the SPI also undergoes severe fluctuations in its receiving and sending time. For this case, the effect is perceived in the acknowledgment and data structure.

A remarkable fact is related to the timing of the CAN and I2C data buses. Both are very similar despite the difference in the number of packets. It means the CAN implementation is almost a factor of two slower to transmit packets than the I2C. A possibility is the higher complexity in the frame generated by the CAN data bus.

Regarding the timing, the RS-422 is the slowest (on average) of all because of its implementation. The stability of the data exchange remains a decisive issue and the overall increase in duration makes it worse. Nonetheless, this data bus has the highest failure rate in the data structure. The surprising part stays the nonexistent error in the measurement themselves despite the issue with the general structure. It shows that some packets might have been lost but the ones received are always correct with no bytes issue.

As already stated, the measurements issue with the I2C data bus is intriguing. Especially since no other data buses seem to be subject to this type of error. By further investing the problem, it looks that some bytes in packets have the probability of being switched. It is a recognized effect since the detection of the packet type represents frequently an issue and some bytes switching is periodically required. Nevertheless, this issue is a key disadvantage for the data

bus since measurement's corruptions can be undetected in the actual PF OBC architecture.

A general desirable behavior among almost all the data buses remains the acknowledgment of the message. As seen in the previous figure, all the data buses (except SPI) have scored 100% regarding the acknowledge packets in both the PL OBC and PF OBC. It means the master has correctly sent its packets, and the slave has consistently received them. The result indicates a robust implementation of the high-level protocol.

An essential point not addressed in this chapter is the size of the data transmitted. It is vital for the PF OBC to receive accurate information from the PL OBC and in particular when the chaser is close to its target. The precision of the data directly influences the quality of the relative navigation of the satellite. It means a high number of bytes should be used to carry measurement. In this study, the limitation has been the maximum size of the packet accordingly to the specification of the data buses. In the future, a method should be developed to overpass this limitation and be able to send more precise measurements over the same packet length. A typical solution would be to split the value into multiple packets.

A consistent decisive issue is regarding the data structure of the reply. By analyzing the data buses, it seems that they all suffer from structure validity failure to some extent. The issue is coming for most of them from missing packets in the transaction. The missing packets are often detected during the transaction but not always addressed. In the I2C and SPI implementations, an extra layer of checks has been developed in this work to reduce this issue. Nevertheless, the increased complexity has not entirely solved the problem because of the instability in these specific data buses. Despite this issue not being on the initial requirement list, it is a key point that needs to be addressed. Indeed, the loss of an entire message means the overall communication would be slower. For example, if the PF OBC request some information and the reply is lost, this computer will reiterate. It results in a reply time about twice as long as initially planned, thus it breaks the timing requirement. Overall, it means another requirement should be established to cover this specific aspect and prevent this kind of issue.

By looking at the various data buses implemented and their performances a first comparison is possible with the SpaceWire/RMAP protocol. The number of bits in average per transaction indicates approximately the data rate of each bus. The analysis of the data buses shows that they are all using about 900 bits of data for one transaction. To be noticed that this represents only the reply of the PL OBC to the PF OBC with a variation between the data buses of 35 bits at maximum. The next step is to divide the bit per transaction of each bus by the average time to transmit. The results are shown in table 2.1. It is important to remember that the data rates shown are computed with the overall transmission time. It means it includes the time between each packet. In addition, all the processes dedicated to encapsulation are not taken into consideration.

The table 2.1 shows that the CAN bus has the highest data rate and SPI the lowest. In all cases, all these data rates are far below the theoretical minimum of the SpaceWire/RMAP with 2.0 $[Mbits/s]$. Nevertheless, it is key to remember that this data rate does not include the time

Data buses	I2C	SPI	CAN	RS-422
Bits per transmission	881	934	896	896
Data Rate [bits/s]	88.1	46.7	99.6	49.8

Table 2.1 – **Data buses summary**

between packets and the overall message encapsulation. It is safe to assume that a practical implementation would be lower than the theoretical one. In any case, this data rate would be still two orders of magnitude higher than the one shown in table 2.1. Generally, this protocol is better suited for intensive data rate when a large amount of data need to be transmitted.

Regrading the data buses tested, the general conclusion of the analysis tends to show the CAN is the best candidate for this backup communication channel. It is a surprising result since the CAN bus has primarily been designed toward a distributed architecture with multiple terminals. Moreover, its architecture includes additional features that might be unneeded for the application described here. Nevertheless, this supplementary complexity might represent the fundamental factor enabling more efficient transactions. Indeed, it has been seen that the data bus is less prone to error while maintaining a relatively high frequency of exchange. A decisive point remains the practically nonexistent fault in the measurement received by the PF OBC. Since the detection of errors in the data structure can be performed by the master, it is less crucial to the overall software architecture. The key implication is the PF OBC would need to request multiple times the same information, however, it is practically guaranteed to obtain the message without error. Nevertheless, it creates an issue regarding the overall transaction timing as described in the previous paragraph. This aspect needs to be worked on if the bus is selected for an ADR mission.

7 Outcome & Incoming Challenges

This chapter introduces the possibility to include a fall-back communication channel in the general architecture of an Active Debris Removal satellite. The decisive point has been to focus on a straightforward but efficient way to support a failure in the main communication bus of the spacecraft. The crucial aspect is to continue transmitting essential data from the Payload On-Board Computer to ensure the success of the mission. This study also brought knowledge of data buses regarding their implementation in a different mission. They might be useful to investigate for less complex ADR satellites. Nonetheless, this protocol aims to carry out critical processed measurements and does not support the transmission of raw data.

The extensive testing of the data buses brought some promising results regarding the implementation of a fall-back data bus. The overall procedure for data verification secures the correctness aspect of the measurement. The message structure possesses the capability to accommodate various types of measurement while transmitting key additional information like timing. The high-level protocol exchange elaborated establishes a baseline for the com-

munication without the need of selecting a specific bus. The data encapsulation and encoding allow for more efficient transmission of information and reduce by almost a factor two the number of bytes needed. These developments permitted the implementation of four different data buses in an Hardware-In-the-Loop testbench. I2C, SPI, CAN, and RS-422 have been the focus of the testing phase. The general conclusion tends to favor the CAN bus over the others because of its prevailing resilience to error. Moreover, its generally fast data exchange and the meeting of the requirements promote it among the other data buses.

To gain a proper understanding of their capabilities, it is critical to move the implementation one step further. A fundamental aspect that should be further investigated is the baud rate or transmission speed of these various data buses. In this first study, the focus has been on their implementation and their resilience to error. The duration aspect naturally constitutes a decisive part, nevertheless, deeper analyses should be conducted with various timing specifications. A significant point to recall is the intrinsic limitation of some data buses to specific baud rates and the link to the internal clock of the hardware boards.

Simultaneously, it would be worth investigating in a sharper analyzing tool, especially regarding the timing. As stated earlier, the limitation of the timing measurement is directly linked to the internal clock of the hardware boards. In addition, Linux based system does not provide a thoroughly reliable clock for sub-second precision. An essential aspect would be to move the Payload On-Board Computer to a Real-Time Operating System and use an external tool to monitor the data on the various buses. This external monitoring would equally benefit the understanding of the packet's failure as well as the quality of the electrical signals.

Overall, these design improvements might be worth for all the data buses tested here and even some additional ones. Nevertheless, the focus on one or two specific buses would allow for deeper analysis and faster testing. In addition, it would enable a finer implementation of the data buses and an early integration into the general flight software of the satellite. Nonetheless, some complement tests are necessary to guarantee a correct selection process. In parallel, the Product and Quality Assurance of the software should be considered. It is decisive to implement procedures early in the development process to ease later verification.

Regarding the high-level implementation, some work could be done on the detection of invalid structures. This failure is prevailing in all the tests and the root is not always clear. By implementing additional processes and checks in the software, a better understanding could be achieved. In addition, it could help mitigate some effects by repairing potential common errors in the packets received. This feature possesses the potential to increase the reliability of the overall high-level protocol interaction.

Design Simulation and Optimization Part II

3 Avionic Simulation

1 Introduction

This chapter is based on the publication "Simulation Tool to Study High Performance Avionic for Active Debris Removal Missions" presented at the Digital Avionic System Conference (DASC) of 2019 in San Diego, CA, USA[152]. It received the "Best of Track" Award.

As for the previous chapter, the content has been modified and extended to better fit the flow of the thesis. The challenge of this first work regarding design simulation and optimization is to get a first step toward the elaboration of complex avionic architectures. It enables to concretize the basis of the simulator and its subsequent developments during the thesis. Most of the assumptions made during the creation of this first simulator are kept for subsequent work. It is important to notice that the tool is not linked to the SimTG simulator mentioned before. In the case of this chapter, a new simulator is developed with an entirely different goal.

One of the essential parts established in this chapter is the basis of an ordinary payload avionic architecture and the physical properties that govern it. In addition, emphasis is already placed on various parts of the simulation that are critical for subsequent developments. In this case, the platform part of the satellite is not taken into consideration and all the features are dedicated to the payload avionic.

This simulation tool has been developed to help design the high demanding payload avionic of an ADR satellite like ClearSpace-1 (CS-1). The simulator provides analyses and trade-offs regarding various hardware configurations. It encompasses the different sensors needed for the mission, the pre-processing part required by some components, and finally the processing/control boards. The latter items can be represented by various architecture types and can contain multiple hardware parts. To reproduce realistic scenarios, the simulator has to consider the requirements of the different mission phases. Notably, they vary in terms of data processing, GNC algorithm complexity, and control loop speed.

To further increase the representativeness, physical models of all components like sensors

or hardware accelerators are provided to the simulator. By combing scenarios and models, an emulation of the data flow is created for a specific system. It includes raw measurements generated by the sensors, the pre-processing operations, and data distribution in the main processing board. The assignment of various tasks like algorithms and control loops at a high-level of abstraction allows the simulator to compute the processor board's usage (or CPU load). On the practical level, the tool uses description files of the various components to create models and table files to generate the scenario constraints. Ultimately, it runs the whole simulation with a defined time step and stores the state of all components at every iteration.

The simulator can provide two primary types of output. The first one is the evolution over time of distinctive architecture's aspects. In this category, there is the energy consumption, the data produced by the sensors, the memory usage, the main processor usage, and some others. It is valuable information when precise analyses of a specific architecture performance are desired. The second type of output is architecture comparison with trade-offs. To accomplish this task, the simulator extracts simulation results from multiple files and compares them to each other. The trade-off matrices are outputs from the simulator and can be adapted depending on the situation. In the end, analyses provided by the tool help classify solutions regarding their suitability to a particular set of requirements. Finally, the user of the tool can then use this information to iterate its design. The code of the simulator can be found in Appendix B.

2 Simulator architecture

The simulator presented has been developed with MATLAB. Its goal is to test and analyze unique configurations of payload hardware for the CS-1 satellite. The simulator is able to load different architecture configurations but also to directly work with mission profiles (Orbit, electrical energy fluctuation, change in tracking algorithm, etc.). The tool facilitates trade-offs between diverse payload architectures of the mission by simulating the data flows, the performance, and the electrical energy required by the different modules. The results are used to develop the RendezVous On-Board Computer (or PL OBC) board dedicated for the rendezvous phase of ClearSpace-1.

The simulator allows the implementation of various architectures types described in the literature review and combines them with the requirements of an ADR mission like CS-1. Moreover, the different GNC and image processing algorithms are represented in a high-level of abstraction to simulate the load of the PL OBC. Ultimately, models of sensors are implemented to simulate the actual data stream.

Figure 3.1 shows the overall architecture of the simulator with the description file in light green and the simulation class in darker green. The simulation is created based on a set of different description files (.xml). These files provide physical descriptions of the different hardware parts and the overall architecture configuration. There are, in addition, table files (.csv) that contain the behavior of the sensors and satellite over time. Each description file can be modified to alter the result of the simulation. Moreover, by modifying the profile of the

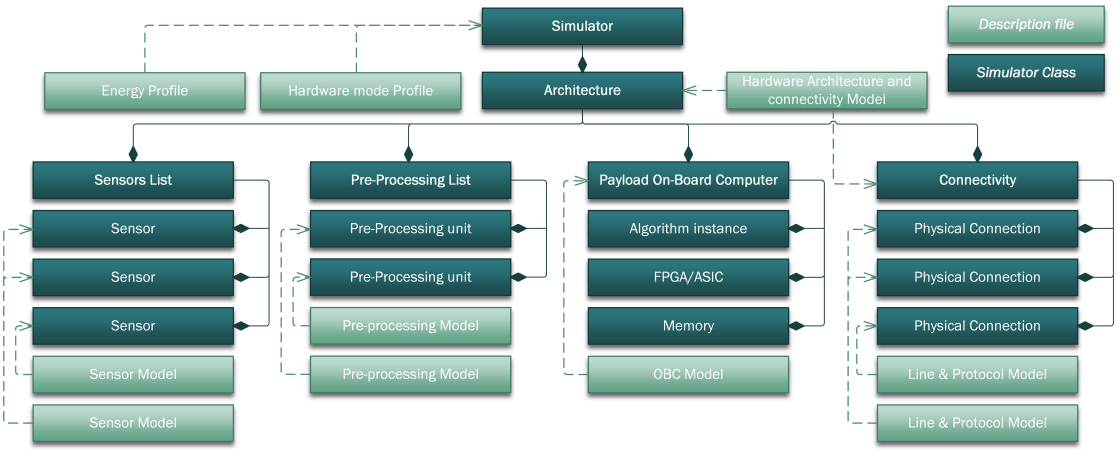


Figure 3.1 – **Architecture of the simulator.**

mission, the architecture's robustness can be tested. As mentioned earlier, all these files can be found in Appendix B.

Concerning the different description files, there are, of course, models of sensors. In these models, basic information is included such as their type and also characteristics of the different modes. A decisive point is to describe with accuracy the output of the sensors depending on their mode. For example, the energy consumption and data rate will be considerably higher in some parts of the mission. Another model is the physical link between elements and the protocol used. Each type of link (LVDS, SpaceWire, I2C, CamerLink, etc.) has a specific data rate, protocol, latency, and also energy consumption. These files convey generic information that can then be applied to the simulation. The following type of model is for the pre-processing unit. Some sensors might need these entities to reduce the amount of data transferred. By describing the internal memory/cache and the type of compression, the behavior of such units is emulated. The most fundamental item is then the description of the PL OBC. The file contains information about the main processor, hardware accelerators, and memories. It equally describes the need for visions algorithms and controllers in terms of computation. The final file is the one describing the general architecture and more specifically the connection between the other blocks.

Part of the simulation comes from mission profiles. These files contain the fluctuation in available electrical energy during the operation as well as the mode changes of the hardware elements. The profile additionally includes the switch between algorithms during the mission. It can be obtained from the literature as test cases or can directly be extracted from the operation simulation of the future ADR satellite.

For the simulation process, the program will read the different description files and load the mission's profile. The following step is to emulate the full payload architecture in the tool and use the profiles to run one part or the entire mission. The simulation does not run in real-time; however, results are stored with all other information about the system over time. The process

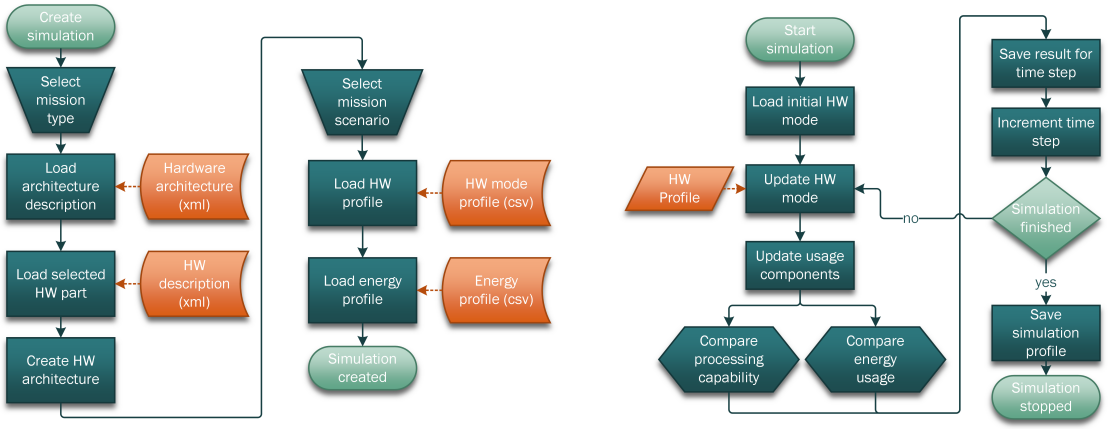


Figure 3.2 – **Process flow.**

flow of the simulation is shown in Figure 3.2.

The core of the simulation works by incremental steps with a defined duration. The first task is to update the mode of every hardware component regarding their specific profile and the actual time step. The second task emulates the data stream inside the system, and it starts with the sensors and then propagates the information up to the Platform On-Board Computer (PF OBC). At every step, the simulator verifies if data packets need to be transmitted or are being sent. In the first case, it initializes the connection with the connected hardware part. In the second case, it simply updates the amount of data that has been exchanged.

The intermediate items on the data stream chain are ordinarily the memories inside the Payload On-Board Computer. They represent the connection between the sensors or the pre-processing parts and the different hardware components of the PL OBC. They are modeled in a high-level of abstraction. To obtain realistic results, it can be considered that these memories have different data rates for their inputs and their outputs.

Once all components have been updated, the simulator takes care of the PL OBC itself. ASICs and FPGAs are taken into consideration first with the amount of data extracted from the memories and the energy they consumed. At that time, the algorithms instantiated on the Payload On-Board Computer are resolved with their respective connection to the distinct memories. These memories are simple elements that stored the amount of information provided by the sensors or the pre-processing unit. At this point, no distinction is done between the various type of data and everything is assumed to be the same. The implementation also makes the memories store all the information received without any analysis. These two design points are further discussed in the comments subsections. On the other side of the memories, the various algorithms are consuming the data. At this point, there is again no distinction in the type of data. It is uniquely assumed that each algorithm requires a certain number of bytes at a defined frequency. In this case, the data used is simply deleted from the memories.

On the processing load, the assumption is done that the PL OBC has some available resources.

On the other side, the various algorithm required a certain amount of CPU load to run which is dependent on their mode. The actual usage of processing load (main core) is simply the addition of all the running algorithms with the exception of the one implemented in the hardware accelerator. It is important to notice that the results are showing the percentage of processing load used. It means that if the CPU is moving to a higher mode, more resources are available and thus the percentage shows will decrease. Finally, there is no limitation on the CPU side if the algorithms are consuming too many processing resources. It means the percentage can technically go over 100%. This design choice is explained later in the chapter.

To summarize, the simulator created provides first-level analyses of different hardware/software architectures. It can load various mission profiles and therefore compares a broad range of scenarios. Moreover, new description files of components are easily added to the simulation, and the remaining ones can be modified. The broad panel of output and the storage of every state of the simulation permit the analysis of various architectures in great detail.

3 Simulation Results

This simulator has been used for preliminary designs and tests. In this section, two payload avionic architectures are presented and analyzed. It is important to notice that these payload avionic architectures are inspired by actual design but mainly simplified. In addition, their parameters are first-level estimations and more work needs to be done to refine them. The main purpose is to demonstrate the capability of the tool.

Regarding the payload avionic, an ordinary design is created with a single PL OBC and several sensors for the first one. The second case describes a more complex architecture with a broad panel of sensors and two pre-processing units. For this initial analysis, three scenarios are applied to these two avionics. These scenarios represent typical phases that could be observed during an ADR mission. As for the payload architecture, the scenarios are representative of specific flight phase and their parameters are first-level approximations.

3.1 Payload Avionic Architectures

1st Case

As explained before, this first architecture can be seen as fairly standard and is shown in Figure 3.3. It is designated as "Av 1" in the results part of the chapter. It contains two sets of dual cameras. One set is dedicated to the far range and the other one to close range. It also contains a laser ranger for the concluding phase of the mission. All these sensors are directly connected to the main Payload On-Board Computer without any pre-processing unit. Several algorithms are running on the PL OBC itself and one of them is implemented on a hypothetical ASIC chip. All the values presented below are estimations of actual hardware parameters and do not represent any specific component.

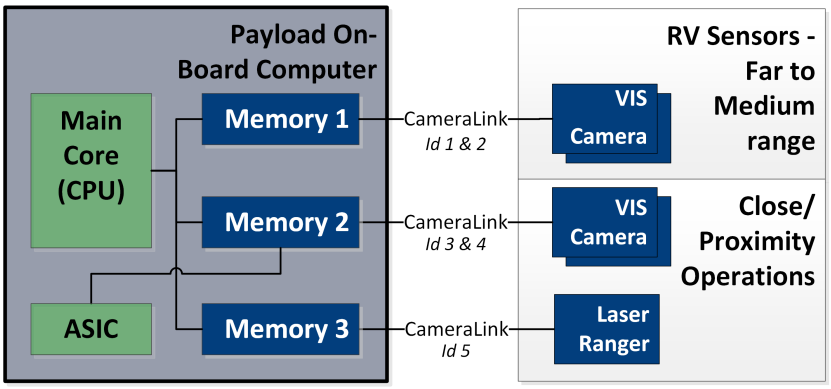


Figure 3.3 – **Payload avionic architecture 1 (Av 1).**

The first set of cameras is for long-range. They possess a wide field of view to first detect the target and then track it. These cameras do not need to produce high-resolution images since exclusively a limited number of pixels could be employed for tracking. Thanks to the extrapolation done on the target attitude, these cameras do not need high frame rates either.

In addition to a medium resolution of 1024×1024 pixels, they have a general low data rate. These cameras can be operated in three various modes. The first one is the Low Energy Consumption (LE) mode. The sensor consumes a reduced amount of energy and provides images at a very low rate. (Energy 10 $[mW \cdot h]$, 5 Frames Per Second (FPS)) The second mode is created to enable target finding operation. It has a higher energy consumption but provides data at a much faster rate. (Energy 750 $[mW \cdot h]$, 66.7 FPS) Ultimately, an in-between mode serves the tracking phase and is also used during the initial approach. It consumes slightly less energy and maintains also a lower rate. (Energy 600 $[mW \cdot h]$, 33.4 FPS)

The second set of cameras is for close-range operation. It possesses a smaller field of view than the previous one and is dedicated to the target's motion reconstruction as well as the constant tracking. These cameras are operated until the final capture of the debris and provide accurate information about the attitude of the object. Moreover, they should provide the data for the motion reconstruction even with a fast spinning target. Ultimately, it means they should have a relatively high resolution of 2048×2048 pixels with also a decent data rate. The first mode is again LE. (Energy 10 $[mW \cdot h]$, 5 FPS) The second mode is dedicated to motion reconstruction. It has higher energy consumption and provides data at a very high rate. (Energy 400 $[mW \cdot h]$, 40 FPS) In conclusion, the tracking mode serves for close proximity operation. (Energy 300 $[mW \cdot h]$, 28.6 FPS)

The laser ranger is here to provide relative distance during the final approach of the target. It allows getting a second set of observations in case of an issue with the illumination of the object or drifting in the image algorithms. The laser ranger is a crucial sensor to prevent any collision with the target. Since it measures a few points, the sensor has a reduced amount of data to transmit. However, the instrument can provide this information at a very high rate. This sensor supports two modes: LE and nominal. Evidently the second provides the data at

a faster rate with equally higher energy consumption. (LE mode: Energy 10 [$mW \cdot h$], Data Packet 0.1 [Mb], Rate 5 [Hz] and Nominal mode: Energy 300 [$mW \cdot h$], Data Packet 2 [Mb], Rate 100 [Hz])

The physical links connecting every element of this architecture are exclusively CameraLink. They allow a relatively high data rate with up to 5000 [$Mbps$].

The PL OBC has, in this case, three distinct memories with a capacity of 4 [GB] each. It embeds as well an ASIC dedicated to motions reconstruction. This component can process raw data at 6000 [$Mbps$] and operate to a frequency up to 45 [Hz]. The PL OBC should additionally include four other algorithms implemented inside its core. There are the "Lost in Space", target tracking (far and close range), and range reconstruction. Each of these algorithms have also four different modes, respectively LE, Fast, Accurate, and "turn-off". The specifications of these modes are shown in Table 3.1. All these algorithms reflect the potential descriptions of an actual implementation. Their parameters are not linked to any specific product.

The ultimate point is about the PL OBC's main core. It includes three unique modes with each time a change in its capability in terms of Million Instructions Per Second (MIPS) as well as in the energy consumption. The LE mode is set to 1000 [$MIPS$] and consumes 2 [$W \cdot h$]. The low-efficiency state has 4000 [$MIPS$] and 6 [$W \cdot h$]. Ultimately, the full mode can go up to 7000 [$MIPS$] and 9 [$W \cdot h$].

Algorithm Name	**Parameters**	**Algorithm Mode**		
		LE	***Fast***	***Accurate***
Lost in Space	MIPS	0.1	1200	1100
	Frequency	10 [Hz]	120 [Hz]	105 [Hz]
	Data rate	200 [$Mbps$]	2600 [$Mbps$]	2200 [$Mbps$]
Target tracking (far range)	MIPS	0.1	900	1300
	Frequency	10 [Hz]	140 [Hz]	110 [Hz]
	Data rate	200 [$Mbps$]	2500 [$Mbps$]	2100 [$Mbps$]
Target tracking (close range)	MIPS	100	2000	2400
	Frequency	10[Hz]	80 [Hz]	70 [Hz]
	Data rate	20 [$Mbps$]	5700 [$Mbps$]	5280 [$Mbps$]
Range reconstru-ction	MIPS	0.1	1700	2100
	Frequency	1 [Hz]	110 [Hz]	90 [Hz]
	Data rate	20 [$Mbps$]	300 [$Mbps$]	200 [$Mbps$]

Table 3.1 – **Algorithm characteristics 1.**

As a reminder, the payload avionic architecture can be seen in Figure 3.3. Each set of cameras and the laser ranger are connected to their own memory inside the PL OBC. Moreover, the ASIC is directly linked to one of the memories.

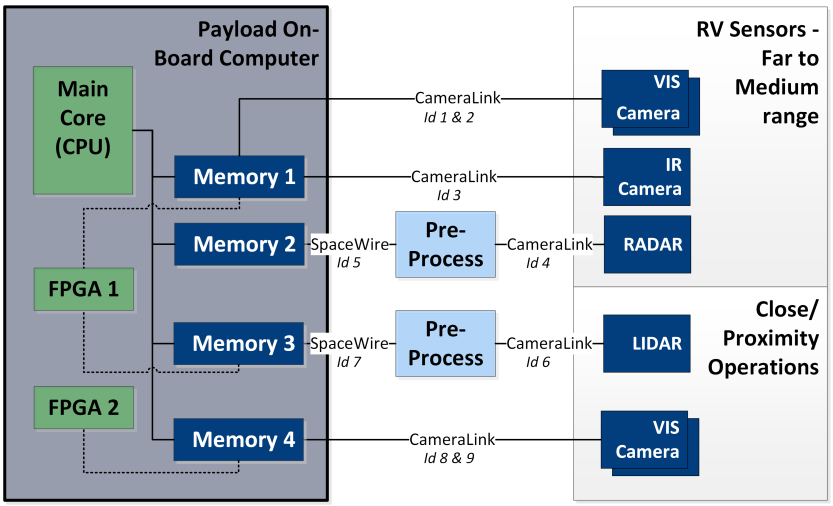


Figure 3.4 – **Payload avionic architecture 2 (Av 2).**

2nd Case

This second architecture represents an upgraded version of the previous one and is shown in Figure 3.4. It contains more sensors and a higher diversity of outputs. It additionally includes more processing power with two pre-processing units as well as two FPGAs in its main core. Its sensors cover a vast range of phases and overlap in multiple areas. Concerning the algorithms, they are similar to the previous architecture with exceptions. As for the previous case, all the values presented are approximations of actual hardware parameters and do not represent any specific component.

The sensors mentioned below go from far range to close range. The first set of sensors is the two cameras dedicated to the long-range. As in the previous example, they possess a wide field of view to first detect the target and then track it. Their resolution is relatively higher than the one presented before (1280 × 1280 pixels). The goal is once more to have frequent updates of the attitude of the target while supporting a reasonable number of frames per second. As in the previous example, these cameras support three modes. The first one is Low Energy Consumption (LE) which consumes a limited amount of electrical energy and has low frames per second. (Energy 10 $[mW \cdot h]$, 2.5 FPS) The second mode is dedicated to target detection with a high data rate and an increase in energy consumption. (Energy 180 $[mW \cdot h]$, 90.9 FPS) The latter one operates for the tracking phase at long range and has a lower frame per second than the previous mode. (Energy 120 $[mW \cdot h]$, 33.4 FPS)

The next sensor in the architecture is the Infrared (IR) camera which enables tracking of the target even in the shadow of the earth. Thanks to the usually limited emission of heat from any object in space, this camera is capable of finding and tracking debris without illumination. Moreover, it has a minor sensitivity to reflection when the target is fully illuminated. This sensor is typically used after the object has been detected and to keep accurate tracking. The sensor has a resolution of 580 × 480 pixels. Ultimately, this camera supports two modes.

The first one is LE with low energy consumption and minimal frames per second. (Energy 10 [$mW \cdot h$], 4 FPS) The primary mode is the nominal one dedicated to target tracking at a medium range. (Energy 400 [$mW \cdot h$], 28.6 FPS)

For the medium range, the spacecraft also has a RADAR. This sensor is a complement to the IR camera for the same distance, but it can provide range. It is resilient against low illumination and is unaffected by light saturation. The downside is higher energy consumption as well as the complexity of the instrument. This sensor also has two modes. The LE one (Energy 100 [$mW \cdot h$], Data Packet 4 [Mb], Rate 2.8 [Hz]) and the nominal one for typical operation. (Energy 700 [$mW \cdot h$], Data Packet 4 [Mb], Rate 20 [Hz])

This architecture (Figure 3.4) additionally includes a LIDAR (LIght Detection And Ranging) instrument. This sensor is typically used to estimate the pose and the rotation of the target at medium to close range. It is extremely valuable since the sensor provides a depth map of the object. The significant downside is the very high-energy consumption as well as the quantity of data per "image". For this reason, the instrument needs some pre-processing parts to reduce the bandwidth. This component is described later. The sensor has three unique modes and the first one is the usual LE. (Energy 10 [$mW \cdot h$], Data Packet 33.4 [Mb], Rate 1.7 [Hz]) The second one is the fast mode which provides less accurate information but at a high rate. (Energy 900 [$mW \cdot h$], Data Packet 33.4 [Mb], Rate 10 [Hz]) The latter one is the accurate mode that outputs precise data at a low rate. (Energy 800 [$mW \cdot h$], Data Packet 33.4 [Mb] Rate 5.5 [Hz])

Ultimately, a set of two cameras is operated for the last meters of the approach. They have a large field of view and are here to monitor the capture. Their role is furthermore to track the debris in the last meters and to prevent any collision with the object. It means they should have a relatively high resolution of 2160×2160 with also a decent data rate. The first mode is once more LE. (Energy 10 [$mW \cdot h$], 5 FPS) The second mode is the motion reconstruction state. (Energy 450 [$mW \cdot h$], 35.7 FPS) The last one is target tracking. (Energy 340 [$mW \cdot h$], 28.6 FPS)

In this architecture, two pre-processing units are used respectively the Lidar and the RADAR. Since these two instruments have a high data bandwidth or complex output, it is critical to reduce and convert the information to prevent overload of the PL OBC. The downside is they create a delay in the data flow and consume energy as well. In both cases, they include modes adapted to the component linked to them. Table 3.2 is the summary of their characteristics.

The physical link connecting every element of this architecture (shown in Figure 3.4) is more complex than in the previous example. The pre-processing units are connected to their respective sensor with a CameraLink interface with 5000 [$Mbps$]. Then SpaceWire with 50 [$Mbps$] is used from the pre-processor to the memories to optimize the data transmission. Ultimately, CameraLink is used with all the other sensors directly connected to a specific memory.

Unit Name	Parameter	Mode	
		Fast	***Accurate***
Pre-processor RADAR	Compression factor	5	7
	Compression rate	250 [$Mbps$]	250 [$Mbps$]
	Lag	2000 [μs]	1000 [μs]
	Energy	100 [$mW \cdot h$]	200 [$mW \cdot h$]
Pre-processor Lidar	Compression factor	6	9
	Compression rate	250 [$Mbps$]	340 [$Mbps$]
	Lag	1000 [μs]	1500 [μs]
	Energy	300 [$mW \cdot h$]	800 [$mW \cdot h$]

Table 3.2 – **Pre-processing unit characteristics.**

The PL OBC has, in this case, four different memories with two at 4 [GB] (number 1 and 4) and two at 32 [kB] (number 2 and 3). In addition, this PL OBC includes two FPGAs dedicated to various tasks. The FPGAs will run unique algorithms depending on the scenario and therefore will be connected to distinct memories. These connections are explained in each scenario. These FPGAs have a limitation in capability, however, this parameter is unused here. It is arguable that the FPGA could be added directly as a pre-processing unit, nevertheless, the simulator has another distinct element for this application. For this implementation, the FPGAs are used as additional hardware accelerators that can be reprogrammed depending on the need of the mission phase. This might not be used in an actual flight demonstration mission, however, the simulator capability is included. The PL OBC includes six algorithms that can be either implemented inside its core or on an FPGA. These FPGAs consume both 900 [$mW \cdot h$]. The algorithms are almost the same as before with "Lost in Space", target tracking (far, medium, and close range), range reconstruction, and motion reconstruction. Each of these algorithms has also four different modes, respectively LE, Fast, Accurate, and Turn-off. It is equally essential to note that if implemented on an FPGA, the processing speed is different. The specifications of all algorithm modes are shown in Table 3.3.

The ultimate point is about the main core of the PL OBC. It supports three unique states with its LE mode having 1000 [$MIPS$] and consuming 2[$W \cdot h$]. The low-efficiency mode has 5000 [$MIPS$] and 10 [$W \cdot h$]. Finally, the full efficiency state can allow up to 8000 [$MIPS$] and consume 14 [$W \cdot h$].

As a reminder, the general avionic architecture can be seen in Figure 3.4. The connections between FPGAs and memories are shown in dashed lines since it depends on the scenario.

Finally, the Figures 3.5 and 3.6 shows the comparison of parameters regarding both avionics. As it can be noted, the second case has a broader range of processing load with respect to the first architecture. The other comment is about the general higher level of consumption for the second avionic. Regarding the frequencies, the first avionic is more consistent among the various algorithms. The second architecture shows two extremes in the variation of the

Algorithm Name	Parameters	Algorithm Mode		
		LE	***Fast***	***Accurate***
Lost in Space	MIPS	0.1	5000	4500
	Frequency	10 $[Hz]$	160 $[Hz]$	120 $[Hz]$
	Data rate	200 $[Mbps]$	3700 $[Mbps]$	3300 $[Mbps]$
Target tracking (far range)	MIPS	0.1	4000	6000
	Frequency	40 $[Hz]$	180 $[Hz]$	120 $[Hz]$
	Data rate	200 $[Mbps]$	3800 $[Mbps]$	3400 $[Mbps]$
Target tracking (medium range)	MIPS	0.1	1800	2400
	Frequency	0.5 $[Hz]$	20 $[Hz]$	1 $[Hz]$
	Data rate	5 $[Mbps]$	50 $[Mbps]$	10 $[Mbps]$
Target tracking (close range)	MIPS	0.1	1900	2600
	Frequency	10 $[Hz]$	60 $[Hz]$	35 $[Hz]$
	Data rate	20 $[Mbps]$	300 $[Mbps]$	200 $[Mbps]$
Range reconstru-ction	MIPS	500	1700	2300
	Frequency	1 $[Hz]$	40 $[Hz]$	30 $[Hz]$
	Data rate	20 $[Mbps]$	400 $[Mbps]$	300 $[Mbps]$
Motion reconstru-ction	MIPS	400	1900	2400
	Frequency	1 $[Hz]$	4 $[Hz]$	3 $[Hz]$
	Data rate	10 $[Mbps]$	15 $[Mbps]$	10 $[Mbps]$

Table 3.3 – **Algorithm characteristics 2.**

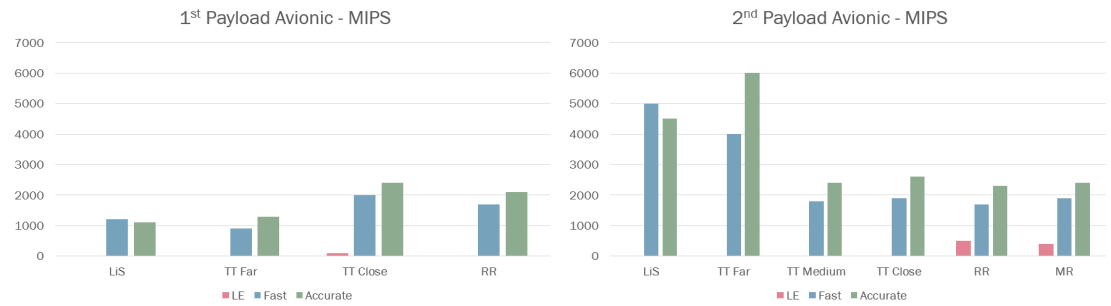


Figure 3.5 – **Algorithms MIPS.**

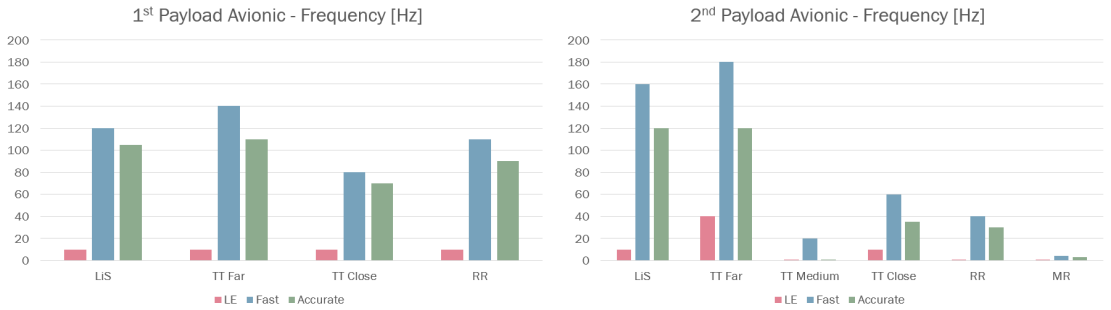


Figure 3.6 – **Algorithms frequency.**

frequencies in comparison with the first case. Nevertheless, additional sensors such as RADAR allow a potentially more precise target recognition.

3.2 Scenarios

In this subsection, three different scenarios are presented with various reactions from the payload avionic architecture. It is important to notice that the scenarios are created based on a hypothetical flight phase of an ADR satellite. The numbers presented are first-order magnitude estimations. Similarly, the behaviors of the payload avionics are based on assumptions that need to be confirmed. The overall idea behind these three scenarios is to test two payload avionics under various constraints without imposing highly realistic flight phases. In the three cases, the duration is arbitrarily set to a few minutes. It allows observing various behaviors without producing a large amount of simulated data.

1st Scenario - Far Range

This first scenario has intriguing results regarding the processing load in its main core. In this case, the flight phase happens when the chaser is a few kilometers behind the target and needs to discover it with its sensors. The scenario has a duration of a few minutes, and it is assumed that the satellite is illuminated by the sun. For simplification, it is assumed to last 7.5 $[min]$ with roughly half dedicated to the finding and the other half to the tracking of the target. On the energy level, the chaser has a constant electrical energy supply since its solar panels are always illuminated during this period. Concerning the mission planning, it begins with the start-up of the far-range sensors and the start of the lost in space algorithm. It is considered that these instruments and algorithms are used until the middle of the phase. At this point, the tracking sensors are switched on as well as the algorithm linked to them. Regarding the algorithm, they are first set in fast mode and then in accurate.

Concerning the operation of the first architecture (Av 1), the two long-range cameras are first set to their target finding mode for about 6 $[min]$. At that time, they are switched to target tracking until the end of the simulation. The PL OBC is switched from a low mode to its full mode after 2.5 $[min]$. It means more CPU resources are available and thus the percentage of usage is decreasing. Finally, the "Lost in Space" algorithm is first set to its fast state for 2.5 $[min]$ and then moved to accurate. As soon as the target is detected, the tracking algorithm is set to the fast mode. Then after a small transition, the "Lost in Space" is put in LE. Almost at the end, the tracking algorithm is switched to accurate and the satellite also turns on the close-range tracking algorithm to initiate the transition.

For the second architecture (Av 2), sensors and algorithms are used in a similar way to in the previous one. The sole exception is the IR camera is turned on when the two long-range ones are set to their tracking mode. In this configuration, the tracking algorithm is implemented on an FPGA.

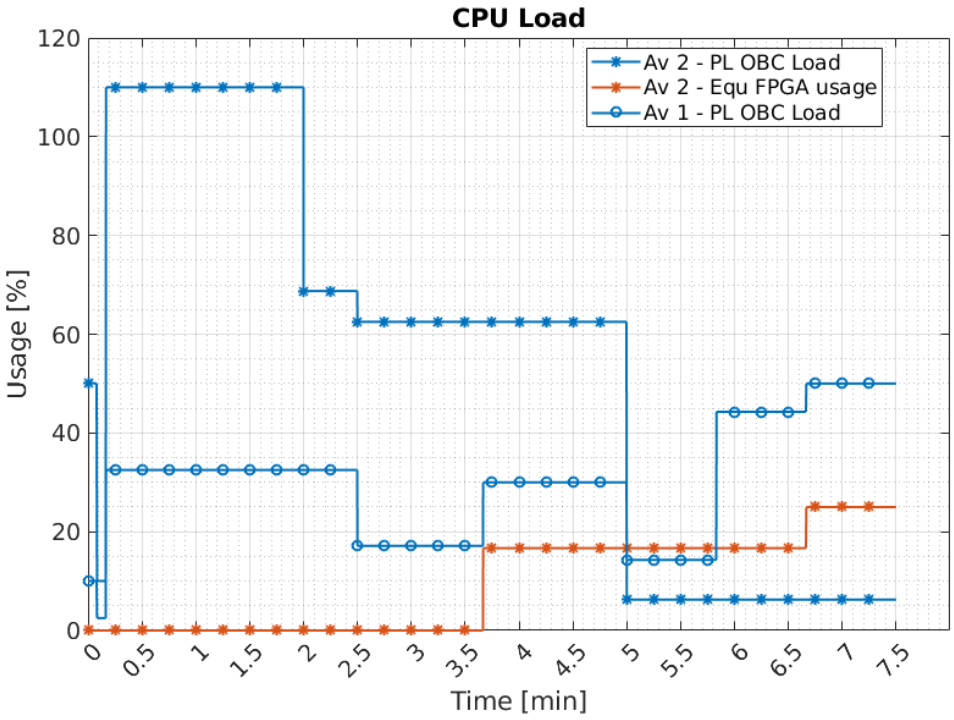


Figure 3.7 – **Scenario 1 - Processing/CPU load.**

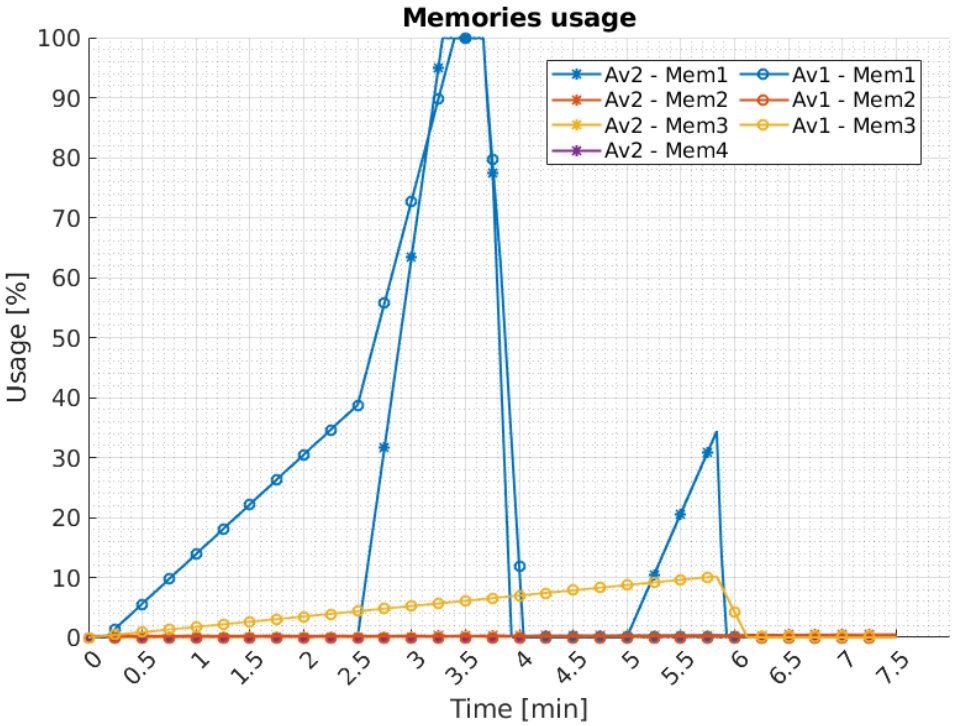


Figure 3.8 – **Scenario 1 - Memories usage.**

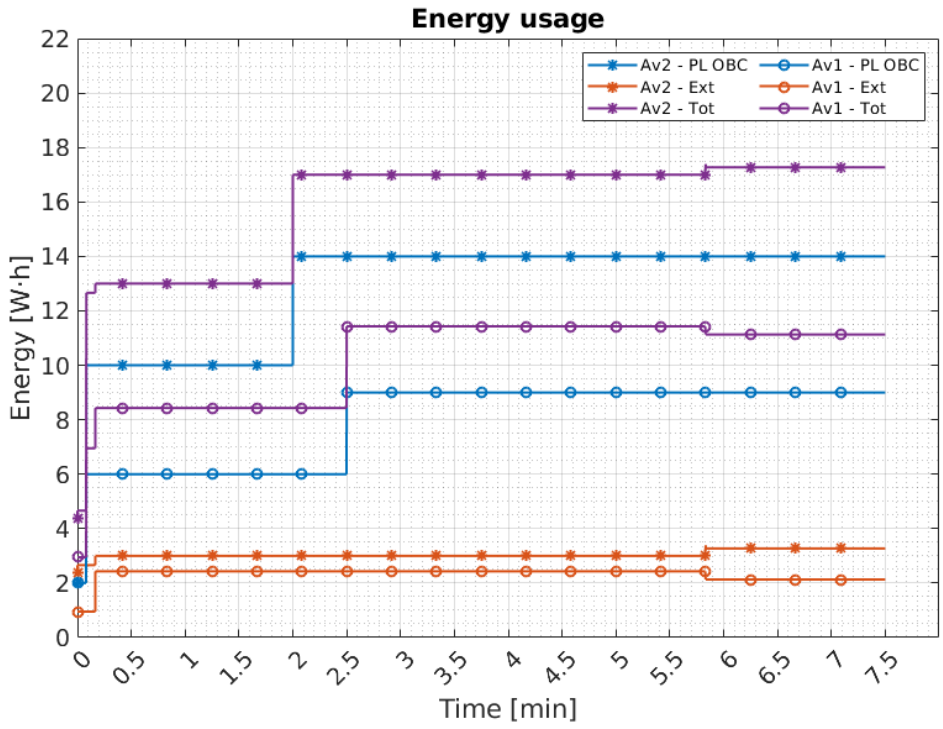


Figure 3.9 – **Scenario 1 - Electrical energy usage.**

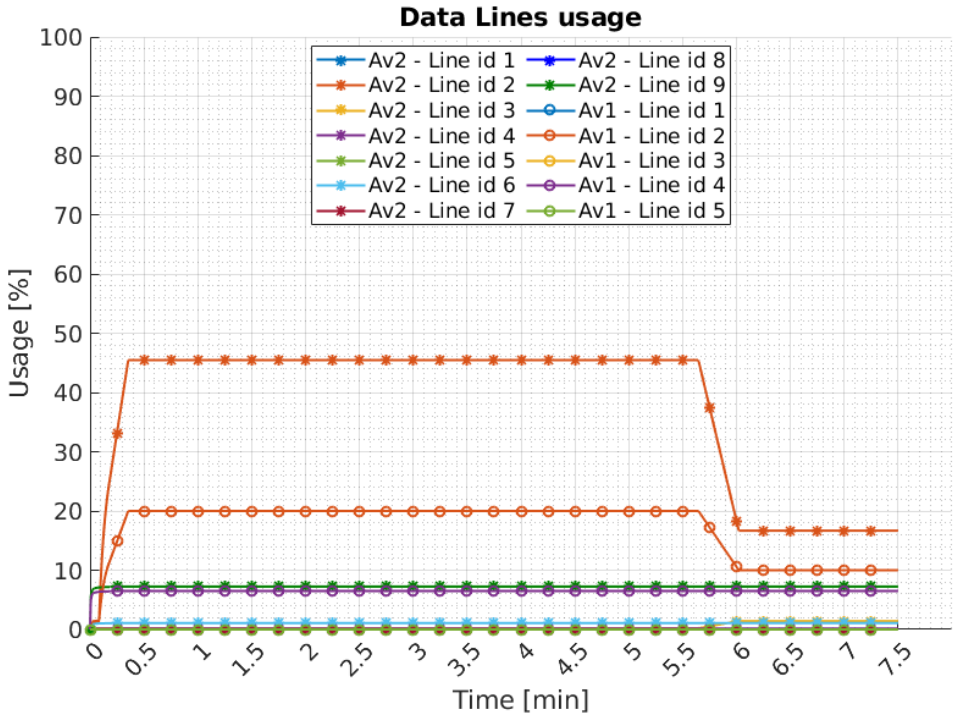


Figure 3.10 – **Scenario 1 - Data line usage.**

In the case of the first architecture, the PL OBC stays busy but never gets overloaded (Figure 3.7). Simultaneously, the memory for the first set of cameras gets quickly full since the algorithm is trying to detect the target (Figure 3.8) and thus is slow. As soon as it is detected, the data are employed rapidly to compute the debris position. There is, in addition, a marginal increase in the third memory linked to the close-range algorithm and camera. Again, as soon as the tracking algorithm is started, the data are completely used. Finally, the energy usage stays always below the available one (Figure 3.9).

In the case of the second architecture, the PL OBC gets quickly saturated and is technically overloaded for about two minutes. Figure 3.7 shows that the CPU load curve goes above 100% at the beginning of the scenario. It is a typical example where the PL OBC should be left on the full mode. In a regular mission planning phase, this behavior would simply not be acceptable and some design change would be required. This aspect is talked about in more detail at the end of the chapter. In the same figure, there is, furthermore, the curve relative to the FPGAs. This curve represents the number of resources moving to FPGAs to release the load on the PL OBC. As it can be observed in Figure 3.8, the first memory is being used above its capability. Because of the intensive rate of the long-range camera, saturation is rapidly reached. This specific behavior is addressed later. Nevertheless, the situation is sorted out as soon as the target is found and the tracking algorithm is turned on. One solution would be obviously to upgrade the size of the memory. Nevertheless, this aspect is discussed later on. Figure 3.9 shows that the energy usage stays always relatively low. Since it is a classical scenario, the result is expected.

In Figure 3.10, it can be perceived that the lines are used predominantly for less than one-fifth of their available bandwidth. The ids of the line are linked to the number written on the payload architecture (Figure 3.3 & 3.4). The sole exception is the second one in the avionic number 2 which reaches almost 50%. To be noticed that this line is linked to the IR camera on the avionic. The general conclusion is the system is not constrained by the bandwidth of the various lines.

2nd Scenario - Proximity Operation

The second scenario is here to describe an intensive phase of the approach which represents the analysis of the target. In this part of the mission, the spacecraft will acquire data regarding the debris at a high rate but for a short period of time. It will iterate multiple times until all the data needed are computed by the PL OBC. This phase of the mission is ordinarily done with a fly-around of the target. It means the chaser will point its sensors to the debris and can not guarantee a constant and optimal illumination of its solar panels. By deduction, it means the available electrical energy is limited to the battery size. For this reason, it can exclusively operate such maneuvers and processings during a short period of time. In this scenario, the chaser attempts one intensive analysis of the target that lasts around 5 $[min]$ with a 2 $[min]$ break before and after.

Concerning the sensors, the spacecraft employs all instruments dedicated to the proximity operation, medium-range and close-range. All the algorithms regarding motion reconstruction and tracking will first operate in fast mode and then in accurate.

For the operation of the first payload architecture, the two close-range cameras are set to their target tracking mode for about 2 $[min]$. At this time, it is the start of the fly-around operation and thus they are set to motion reconstruction mode. The laser ranger is also turned on and stays in a nominal state during the whole phase. Just before this, the algorithms for target tracking and range reconstruction are moved from fast to accurate mode. After 5 $[min]$ of intensive operation the algorithms and the sensors are set back to a less demanding mode. During the whole operation, the ASIC with the motion reconstruction routine is running. Concerning the PL OBC, it activates the full mode during the fly-around otherwise it stays in a low state.

For the second payload architecture, the operations are slightly more complicated than for the previous one. In this configuration, the chaser uses almost all its sensors as well as its two pre-processing units. For the first minute and a half, the two close-range cameras are placed in their tracking mode, and all the algorithms (except "Lost in Space" and long-range tracking) are set to their accurate state. When the fly-around starts, the whole assortment of sensors except the long-range cameras are switched to their most efficient mode, and the pre-processing units are put to the full mode. Regrading the algorithms, uniquely the motion and range reconstructions are set to a fast mode. The others stay in the same state. At the end of the fly-around, the Lidar and RADAR are put in their low mode, and both reconstruction algorithms are in their accurate mode. In this scenario, the short-range tracking and the motion reconstruction are implemented on FPGAs. Ultimately, the PL OBC is set to a full mode one minute before the fly-around and to a low mode one minute after the end of the maneuver.

In the case of the first payload architecture (Av 1), the PL OBC is used to almost its full capability. Nevertheless, there is a narrow margin during the fly-around since it switches to a more powerful mode (Figure 3.11). The consequence is an increase in the electrical energy consumption that can be seen in Figure 3.13. By looking at the memories, it can be observed that the fly-around phase produces many data and the tracking algorithms struggle to keep up with the rate. As soon as the maneuver is finished, the algorithms quickly catch up with the data (Figure 3.12).

In the case of the second payload architecture (Av 2), many resources are being used by both the PL OBC and the FPGAs. Since part of the workload is moved to these FPGAs, the PL OBC still maintains margins (Figure 3.11). Figure 3.12 shows the amount of data produce by the RADAR and LIDAR inside their memories. However, it has to be remembered that these two memories are significantly smaller than the others. Thanks to the pre-processing units, the amount of data can be maintained to a reasonable level. As stated before, the behavior is discussed later. Figure 3.13 shows that the energy usage goes to a high level during the decisive

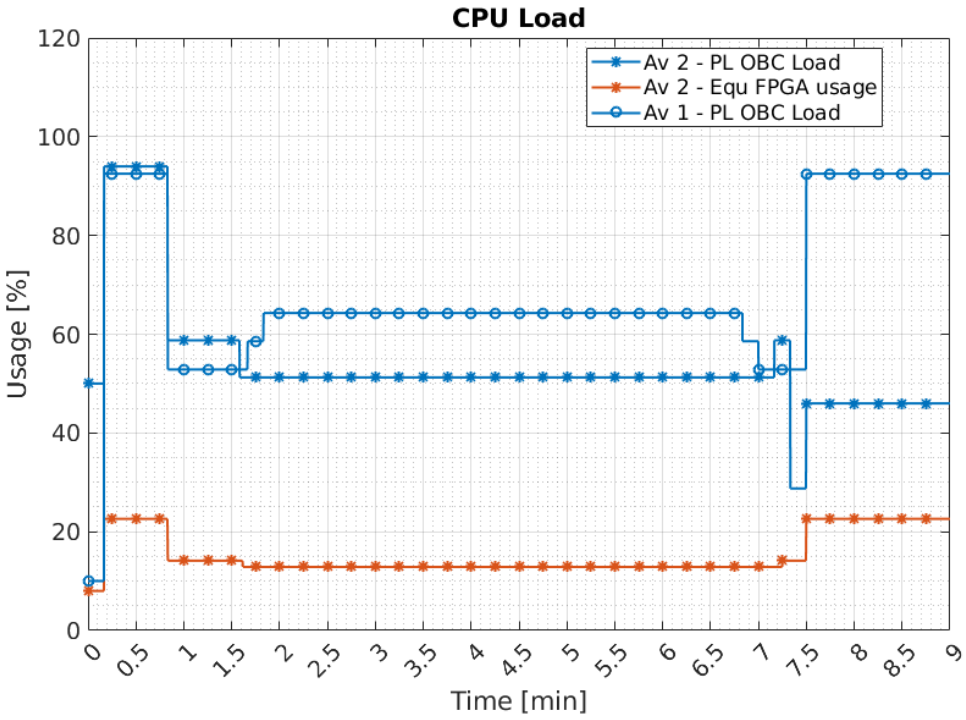


Figure 3.11 – **Scenario 2 - Processing/CPU load.**

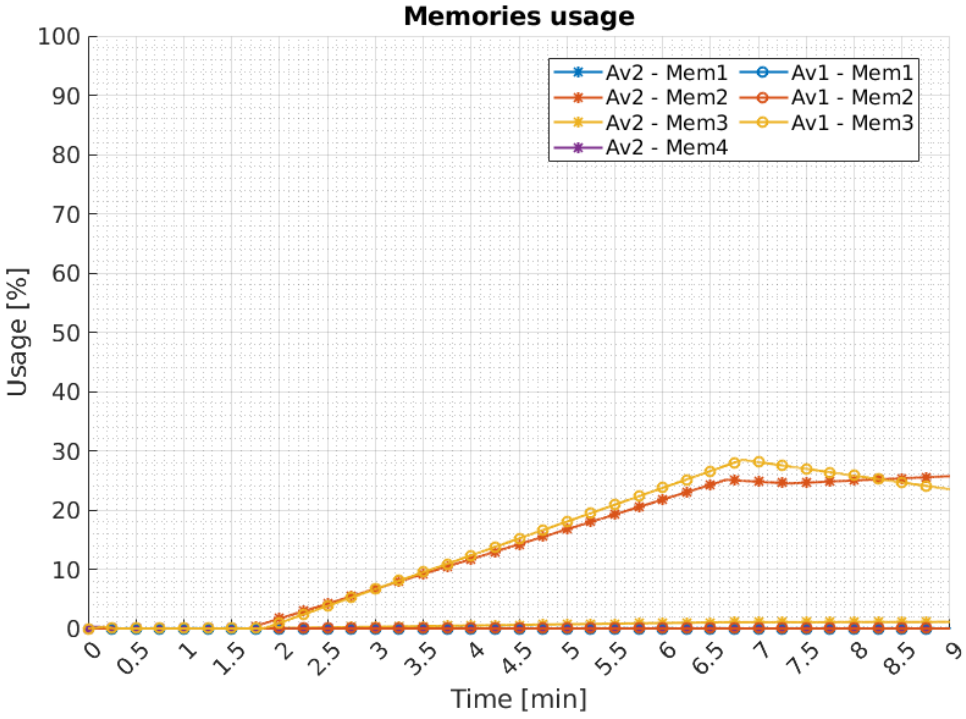


Figure 3.12 – **Scenario 2 - Memories usage.**

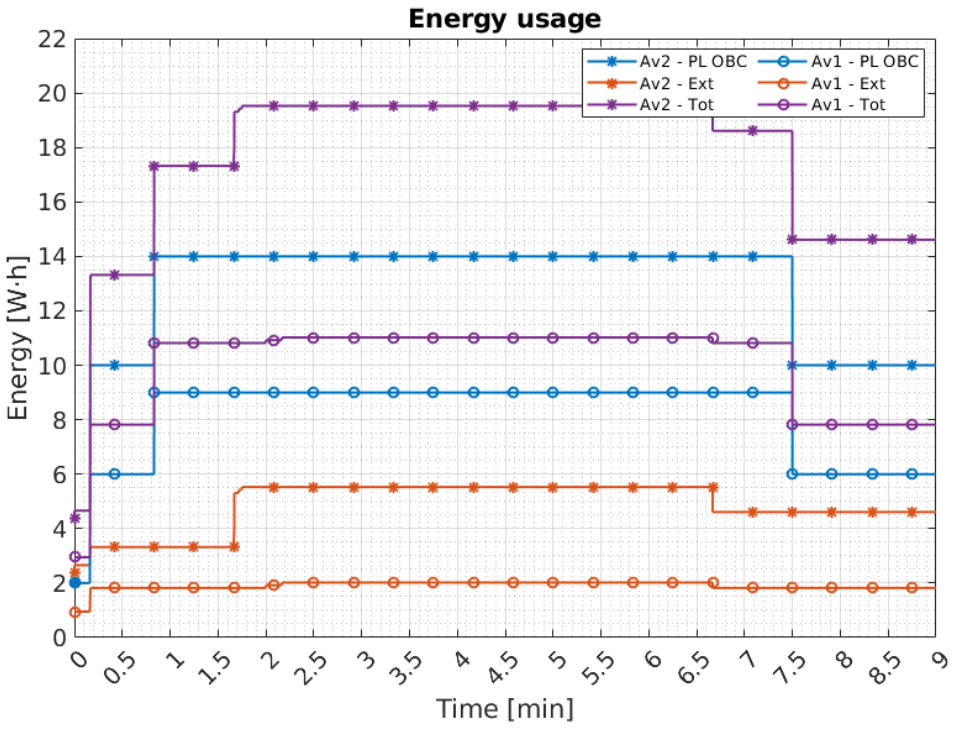


Figure 3.13 – **Scenario 2 - Electrical energy usage.**

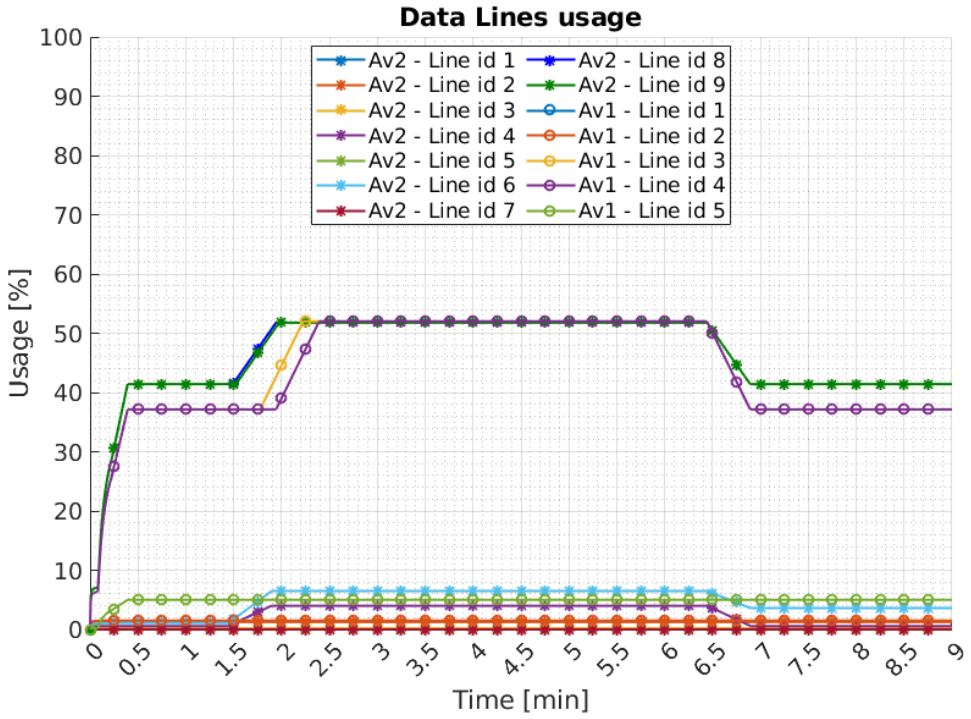


Figure 3.14 – **Scenario 2 - Data line usage.**

phase of the fly-around.

In Figure 3.14, it can be perceived that the lines of the first architecture are pretty busy but there is, nonetheless, some margin. Regarding the second avionic, the lines are used predominantly for half their available bandwidth. Interesting enough is that the ones going out of the pre-processing units are barely used thanks to the compression. Once more, the lines are not a constraint for the system in both cases.

3rd Scenario - Capture Phase

This third scenario represents the final approach of the chaser toward the target. It is the ultimate step before the debris is captured and will last around 5 $[min]$. In this mission phase, any collision with the target must be prevented. Moreover, the capture system will need to know at every moment the pose and rotation of the debris to catch it safely. All the operations must be carried out at a decent rate and with a high-level of accuracy. In other terms, it is important to know very precisely where the target is at all times. Nevertheless, the refresh rate is less critical since the relative velocities of the target and the chaser is slow. It means it is impossible to have a fast (or sudden) change of motion on the debris side if the velocity and rotation of the chaser are set correctly. In parallel, all sensors that can not provide pertinent information are set to LE mode.

The second key constraint of the mission phase is the energy limitation. Again, the chaser needs to be oriented toward the target and it is probable that the solar panels can not receive the optimal illumination. In summary, this phase implies limited available energy linked to the batteries' size and an intensive data rate of various sensors. It is as well needed for the algorithms to compute information as accurate as possible to precisely catch the target without breaking it.

For the operation of the first payload architecture, one of the close-range cameras is set to its target tracking mode and the other to motion reconstruction. The laser ranger is switched to a nominal mode as well. All these sensors are put in LE after the capture, about 8 $[min]$ later. The PL OBC is activated directly into its full mode and stays in this state for the entire operation. The ASIC is on top set to its full mode with a motion reconstruction algorithm for the entire simulation. The range reconstruction and the target tracking algorithms are first set to an accurate state for about 5 $[min]$. After this period, they are switched to their fast mode even though it is not required.

For the second payload architecture, exclusively the two close-range cameras and the LIDAR are used. One camera is set to tracking mode and the other to range reconstruction until the end of the capture 8 $[min]$ later. The LIDAR is switched to its fast mode for a similar period of time. In the scenario, the chaser uses the close-range tracking algorithm as well as the range and motion reconstruction. The first two are implemented on FPGAs. The motion reconstruction is put in an accurate mode until the capture when it is set back to LE. The two

others are first used in an accurate state for 5 $[min]$ before switching to their fast one. The PL OBC stays in full mode for the entire capture phase.

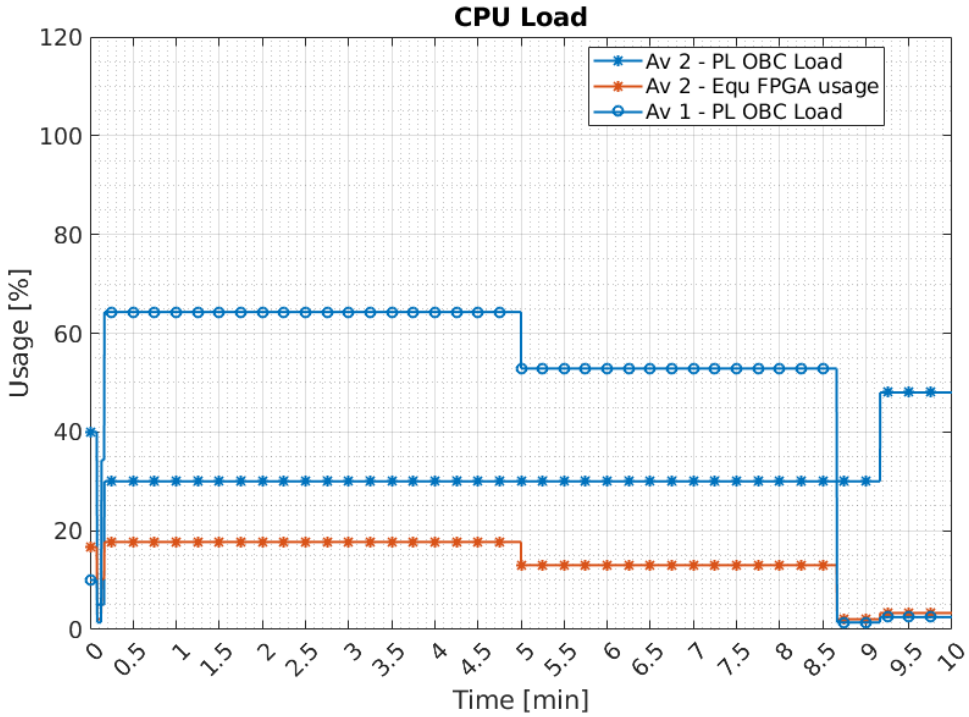


Figure 3.15 – **Scenario 3 - Processing/CPU load.**

In the case of the first payload architecture (Av 1), the PL OBC is again used to no more than 50% of its capability, meaning there is still a margin for a more complex algorithm (Figure 3.15). Even when on a full mode and with all the different sensors, the energy consumption is fairly high. As for the previous scenario, there is a decrease in available electrical energy due to the orientation of the satellite toward the target and not the sun (Figure 3.17). As seen in Figure 3.16, the memories have difficulties keeping up with the intensive data rate, especially regarding the two close-range cameras. The saturation of the second memory will introduce a loss of data and probably induce error in the computation. As seen, the change in the algorithms permits the chaser to return to a nominal state. Concerning the third memory, the constraint is less significant since the laser ranger produces fewer data.

In the case of the second payload architecture (Av 2), the situation is the same as for the scenario 2. Figure 3.15 shows that the PL OBC is not used as its full performances. Moreover, the FPGAs are taking part in the workload. In Figure 3.16, the spike in memory 4 corresponds to the two close-range cameras producing a large amount of data and the algorithm behind having difficulties maintaining the rate. As soon as the cameras switch modes, the motion reconstruction algorithm can process all the remaining information. As in the previous scenario, the electrical energy usage goes to a high level (Figure 3.17). The reason is once more the imperfect alignment of the solar panels relative to the sun. Because of the chaser being

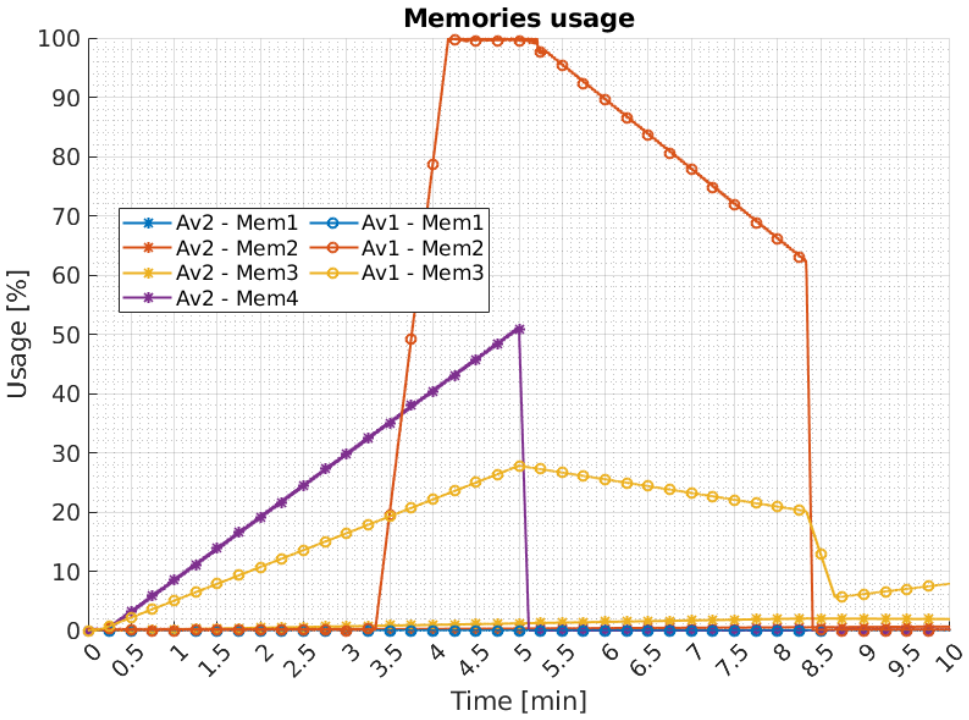


Figure 3.16 – **Scenario 3 - Memories usage.**

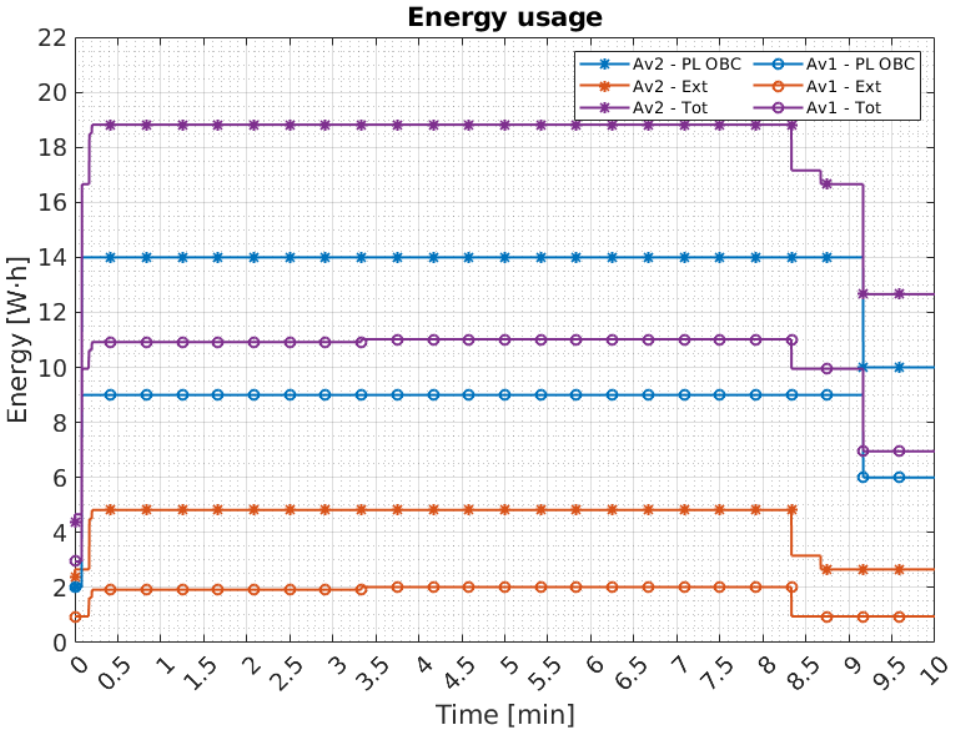


Figure 3.17 – **Scenario 3 - Electrical energy usage.**

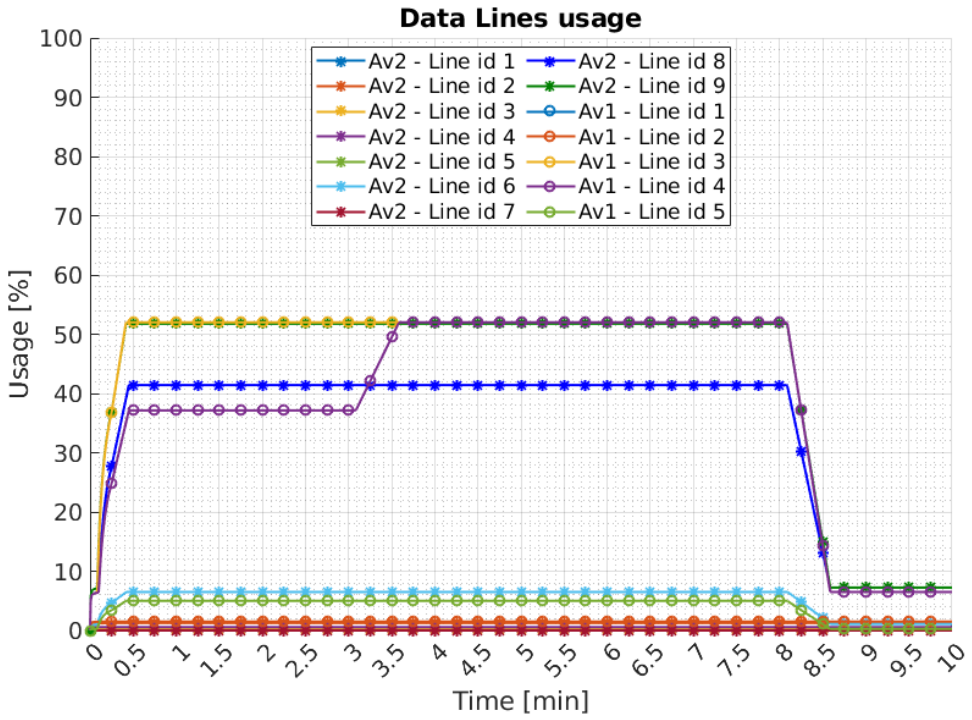


Figure 3.18 – **Scenario 3 - Data line usage.**

pointed at the target. Since the chaser has to point its sensors toward the target, it is impossible to use the panels at their full efficiency and thus the electrical energy in the batteries decreases rapidly.

In Figure 3.18, the lines of the first architecture are not used to their maximum capacity and remain at a ceiling of around 50%. Regarding the second case, it can be noted that they are never overloaded and few stay above the 50% mark. As for both of the previous scenarios, the system is not constrained by the line bandwidth.

3.3 Comments

These three unique scenarios have put the two architectures in various situations. Even though they are both different in their algorithms and sensors, it presents an adequate approximation of their respective performances. As mentioned earlier, the scenarios and architectures shown do not reflect any real design but are here as a proof of concept for the tool. Nevertheless, it is interesting to analyze the results outputted.

Concerning the first scenario, the second architecture imposes more constraint on the PL OBC especially at the beginning in comparison with the other avionic (Figure 3.7). This behavior comes from the difference in the "Lost in Space" algorithm which is more demanding in the second case. Despite this fact, the load of the seconds avionic PL OBC (main core) is rapidly

reduced as soon as it is shifted to a higher mode. Moreover, the FPGA used in the second case take part of the processing load from the main core. Energy-wise, both architectures have a low consumption even though the second one is on the conservative side. Figure 3.8 shows the similarity in the memories behavior. In both cases, there is saturation due to the "Lost in Space" algorithm. This type of result indicates a lack of performance regarding the algorithm bandwidth.

The second scenario is an interesting comparison case since both architectures have a contrasting configuration. In this case, the PL OBC of the first avionic is practically at saturation at the beginning and then again at the end of the fly-around. Despite having less intensive algorithms than the second architecture, the entire load goes directly to the PL OBC. In comparison, the second one stays almost at 50% during the entire operation (Figure 3.11). In both cases, the memories are gradually getting filled by all the sensors, however, the algorithms can maintain the pace. Finally, on the energy level, the second architecture is consuming a high level of energy. For the first avionic, the architecture is less complex therefore limiting the energy consumption of the various units (Figure 3.13). For the second one, the fly-around phase imposes a higher load on the sensors and algorithms which increases the consumption.

The third scenario again implies a broad panel of sensors for the second avionic. Despite this fact, Figure 3.15 shows that exclusively a small fraction of the PL OBC is used during the operation of the second architecture. In comparison, the first avionic goes up to 65%. The FPGAs are a considerable help for the second case since they discharge loads on the PL OBC. As stated before, the energy consumption in the case of the second avionic goes to a high level due to the number of parts consuming energy. This problem can be leveraged by reducing the energy consumption of the PL OBC. Indeed, the PL OBC is under-used in the second avionic, meaning its performance can be lower to conserve energy (Figure 3.17). This problem does not appear on the first avionic thanks to the limited number of parts.

On the general side of all scenarios, an argument is regarding the speed or frequencies of various algorithms and sensors. In these test examples, the tendency has been to set these parameters to a relatively high level. For example, a tracking algorithm at a far range would probably never need a frequency of 180 $[Hz]$ for its operation. Even though it is some parameters in the simulation, it reflects the general high-level of model abstraction used and the need for refinement. As explained, the goal of this chapter was to demonstrate the capability of the tool without defining highly accurate models.

On the memories level, Figure 3.16 shows that the second architecture has less struggle than the first one and mainly thanks to the pre-processing units. In addition, the algorithms in the second case are more efficient, and they are capable of maintaining the rate with the sensor's bandwidth. Despite the increase in performance, the effect is not perceived on the processing load in the CPU since this avionic is using FPGAs. An interesting fact about the usage of the line is that both architectures follow roughly the same curve even though the second one has many intensive sensors. This effect comes again from the pre-processing units available in the

second avionic. In any case, the choice of keeping all the information produced by the sensors in the memories is arguable. A better design would be to keep uniquely the most updated version of it and discard any old parts. A compromise will be to save some key data over a certain period of time if no algorithm is requesting the information. Overall, one solution could be to simply upgrade the memories to have a high storage capacity which would ease the constraints. This general behavior would highly relax the constraint applied to the memories.

Predominantly, observations can be made that the first avionic is usually more loaded in terms of processing power than the second one. In all these scenarios, the first PL OBC has to handle almost all the algorithms of the satellite and it uses most of its resources. In comparison, the second avionic is generally less busy since the FPGAs are taking part of the load on their own circuit. In the first case, an ASIC is being used but due to its determined nature, it is solely valuable for part of the mission. On the opposite, the FPGAs are re-programmable and can serve multiple applications throughout the entire mission. Even though the change of design is time-consuming and less reliable, risks are being mitigated. Switching FPGA programs can be done during less demanding phases or even station keeping at a safe distance from the target.

The primary drawback of the second architecture is its energy consumption. With the accumulation of sensors, pre-processing units, and FPGAs, the satellite want often to consume a large amount of energy that might not be available at the satellite level. One mitigation could be to increase the number and size of the solar panels but it creates other constraints at the system engineering level. Another solution could be achieved by implementing less demanding hardware by reducing performance.

Another restriction remains the type of line used in both architectures. CameraLink is something immensely valuable because of its high data rate, however, it is not a type of connection designed for space applications. The link could potentially constitute problems. The fundamental one would be the reliability of the line. Nevertheless, it can be tackled by implementing a higher redundancy or using other space-grade lines.

On the general usage of the tool, it is important to understand its limitations. The tool has been designed to help mitigate the design phase of ADR satellites such as CS-1. Nevertheless, it does not provide ground truth or an optimal solution. Its goal is to assist the engineers in designing the payload avionic or doing the mission planning. It is the responsibility of the tool's user to understand correctly the output of the tool and iterate the design accordingly. As an example, the energy consumption is exclusively shown at the payload level in the plots. It is primordial to assess the current consumption of the entire satellite and analyze its consequences on the payload avionic. As a second example, the processing resources or CPU load is a key part of these simulations. Nevertheless, it is shown that on multiple occasions the consumption goes close to or over the available resources. In any space mission, this behavior would not be acceptable and requirements would impose a higher limit. In this aspect, it is a typical case where the engineer using the outputs needs to assess this information and modify the design

accordingly.

As a final remark, the tool does not consider any real data fusion such as Kalman filter. The assumption made is that an algorithm can use data from multiple sensors without any distinction. This behavior is at a high level of abstraction and refinement need to be done. In a similar aspect, the FPGAs are exclusively high-level components that would need more granularity to be representative.

4 Outcome & Incoming Challenges

This simulator has allowed preliminary studies on various payload architectures. By comparing performances of hardware and software implementation with respect to the mission scenario, trade-offs and design choices can be made. This tool also enables the validity assessment of various requirements both at the algorithm and hardware levels. The multiple results of the simulator provide insight into a range of avionic parameters.

By monitoring the bandwidth in every line of the payload architecture, optimization can be performed to ensure the reliability and the performance of the connection. Moreover, it provides knowledge regarding the data produced by the various sensors. The analysis of processing load in the CPU remains a rapid way to detect if algorithms are consuming too much processing resources or if their capabilities can be improved. The monitoring of the memories represents, furthermore, an accurate method to check the data management inside the PL OBC and the various accelerators. Even though the concept behind the memories could change, it provides precise information regarding the time required by the sensors to fill them to their full capabilities.

The simulator created still needs some improvements in terms of models, verification, and capability. The first feature is to refine the simulation parameters. In this category, there are improvements to the models of the sensors and the other parts. For this, representations of existing hardware need to be implemented by employing the analysis carried out in the previous part. In the model category, there is, in addition, the PL OBC, the hardware accelerator, and, of course, the memory. By providing realistic models of these components, a more accurate simulation could be achieved. As explained earlier, there is also the GNC and image processing algorithms needing to be improved.

Another decisive part to refine is the interaction between these various models. The first version of the simulator uses oversimplified models for the line and the protocol. Characteristics like clock synchronization, encapsulation, and lag are targeted. By enabling a more advanced representation of these interactions, improvement in the quality of the simulation is expected. In addition, the principle to store all the information in the memories is a design choice that might need to be updated. Finally, the sensor fusion part needs to be addressed. In this first modeling, this concept was not considered and all the sensor data are considered equivalent. A required addition would be to implement a high-level process to simulate the data fusion

and thus have a more representative payload avionic architecture.

In terms of other future work, the validation of the simulator and models is a capital point. By comparing the results and behavior of the simulation with real hardware implementation, crucial information on the validity of the program can be obtained. The goal would initially be to analyze the results with Commercial Off-The-Shelf (COTS) sensors like cameras and small processor boards.

By implementing openly available image algorithms and some basic controller on a COTS processor, a comparison with the simulated version could be performed, therefore providing an initial validation. The following step will be to integrate the FPGA and other hardware accelerators in the preliminary setup to further validate the simulator. The validation can be pursued along with the development of the Payload On-Board Computer of CS-1 by implementing models of the preliminary design either be hardware part or algorithm. Moreover, the tool will keep providing information about the performance of the PL OBC unit during its development.

Finally, the last major point is the overall optimization of the whole architecture. In this initial chapter, the simulator is fixed to a defined architecture and is reacting to input previously defined. To move to more resilient and flexible outputs, it is required to apply some mathematical optimization. The topic of the following chapters is the implementation of an optimizer for payload avionic architectures.

4 Avionic Optimization

1 Introduction

This chapter is based on the publication "Simulation Tool : Resources Management in High Performance Avionic for ADR Missions" presented virtually at the IEEE Aerospace Conference of 2021 in Big Sky, MT, USA[153]. Part of the content has been modified and extended before being added to the thesis. The challenge described in this chapter is to move from a pure simulation to an optimization tool. The primary task is to take all the equations and physical representations of the simulator and convert them into constraints for the optimizer. The end result is a tool able to provide trade-offs of payload avionic architectures for future Active Debris Removal (ADR) space missions. An important point is that the goal of the optimizer varies from the simulator presented before.

This new version of the simulation tool is designed to optimize the resources management inside the payload avionic architecture. The simulator has to decide the set of instruments to use in conjunction with the types of algorithms to deploy. The goal is to find the optimum set that satisfies the physical constraints like available energy and limit in the processing resources as well as requirements of the mission with mainly the accuracy on target tracking. This problem is solved using an Optimal Control (OC) approach with a Mixed-Integer NonLinear Program (MINLP) definition.

This tool has the task to optimize the similar payload architecture that was shown in the previous version. Nevertheless, new payload avionic architecture are suggested to better fit the ability of the tool. To control the overall architecture design, the optimizer includes multiple input and states variables. In this category, there are the statuses of the various sensors. Either they are turned on or off. There are, in addition, the states of the sensors. As in the previous version of the simulator, the sensors support distinctive modes that represent their energy consumption as well as the amount of data they produce. The two other types of input variables are the modes of all the algorithms and the state of the Payload On-Board Computer (PL OBC). Overall, these modes are altering the general behavior of these diverse parts.

Regarding state variables, they can be divided into two categories. The first one is the physical parameters of the system with the energy consumption, the processing resources (or CPU load) available, the data rate at the PL OBC, and the storage capacity. In the abstract one, there is uniquely the information or accuracy on the target. It is a representation of the validity of the data produced by the algorithms. The higher the parameter is, the better the confidence in the information is. In an ideal situation, this parameter would stay at its maximum during the whole mission.

The following section describes the purpose and the exact implementation of the optimizer with a MINLP definition as presented in the introduction chapter. The next step is to demonstrate the capability of the tool with examples of payload avionic architecture. As for the previous chapter, the architectures and the scenarios are inspired by ADR mission but do not reflect any actual design.

2 Optimizer Implementation

This section is dedicated to the problem definition and the implementation of the system into a MINLP solver. The tool employed for the optimizer is the CasASDi toolbox [167] implemented in MATLAB. The advantages of this toolbox remain its ability to solve a considerable variety of optimization problems and the ease of system implementation.

As it has been described before, the goal is to optimize resource management inside the future payload avionic of CS-1. In comparison, the previous chapter was looking at the general behavior of a payload architecture with predefined scenarios and mission planning. The aim of this tool is to provide potential design for a payload avionic architecture. In addition, it outputs a suggestion regarding the mission planning at the payload level for specific scenarios. To obtain such results, the optimizer has the task to select the mode (or behavior) of all the systems elements over a specific duration. This mode selection allows it to manage indirectly parameters like the energy consumption or the CPU load. This approach has some disadvantages that are discussed at the end of the thesis.

For this problem, it is assumed there are a number of sensors on the satellite as well as algorithms that can be run in the main payload computer (PL OBC). Each of these elements supports a set of modes associated with them, and they define their various parameters. For example, a camera sensor could include three unique states that represent a Low Energy Consumption (LE), a fast, and an accurate mode. In this example, it could be imagined that the fast one supports a larger number of frames per second than the accurate one. Nevertheless, the quality of the image would be worse. These modes also influence the electrical energy consumption and some other parameters. To ease the development of the optimizer, it has been decided to keep all the elements at a high level of abstraction. It means that all the elements have a defined set of parameters that can represent any specific component. This drawback of this approach is discussed later in the chapter.

For the practical implementation of the optimizer, a direct approach with Multiple Shootings is adopted. The simulation has K steps and a time interval of t_0 to t_f. It gives an increment of $dt = (t_f - t_0)/K$. This increment is set to observe the reaction of the system with precision. The optimizer uses a Runge-Kutte 4 Integrator to evaluate the ODE (Ordinary Differential Equation) at every step. Ultimately, the optimizer uses the "BONMIN" solver[145].

In all these simulations, the critical state remains habitually the "information on the target" or accuracy of the algorithms. It is the major driving factor of the electronic and primary purpose of the satellite for many of its mission phases. The other states are still relevant, but they are mainly implemented as additional constraints.

2.1 State and Input Variables

This problem follows the general formulation of a MINLP with some input and state variables, the dynamics, some constraints, and finally an objective function. In the case of this system, the problem has five main state variables that are called "X" and are all time-dependent.

The variables presented below are driven by the characteristics of the avionic. Since the goal is to develop a tool that needs to be representative of the real world, the first task was to select the physical properties of the system. Naturally, it will be overly complex to select all of them, and a choice has been made. The initial guess was to focus on the more typical properties that define most of the system elements. To simplify the implementation, it was decided to choose uniquely one abstract variable that represents the overall aptitude of the system. All parameters are explained below in the text.

- $X(1) = c(t)$: Information on the target or accuracy (scale $[0;1]$).
- $X(2) = e(t)$: Electrical energy available.
- $X(3) = r(t)$: Processing resources or CPU load available.
- $X(4) = d(t)$: Data rate at PL OBC.
- $X(5) = s(t)$: Memory storage capacity usage.

In this formulation, the first state variable $c(t)$ is representing an abstract measurement of the system accuracy regarding the target detection. It has been decided to restrain this variable between 0 and 1. The higher the parameter is the more confident the avionic is about the shape and motion of the target. For example, an algorithm could be outputting the pose estimation of the target. In this case, the $c(t)$ variable represents the confidence of the algorithm about its computed result. The contribution of the algorithm to this theoretical value has to be decided by the user of the tool. Nevertheless, it is a convenient way to simplify the complexity and variety of the algorithm. In other terms, it detaches the optimizer from the actual implementation of a specific algorithm. The validity of this assumption is discussed

later in the chapter. About the algorithms, they all represent potential processes dedicated to an ADR mission. It means providing information to the spacecraft about the attitude and position of the targeted debris. They can be image processing or GNC algorithms. Similarly to the accuracy of these algorithms, the user has to define their CPU usage as well as their frequency. Evidently, it is not a trivial task to achieve, and thus first-hand approximations are used in this chapter.

All the other state variables represent the physical parameters of the system. The first one $e(t)$ represents the energy consumption on the payload avionic side over time. $r(t)$ is the CPU load available in the PL OBC for the algorithms. The next one is $d(t)$ which represents the data rates of all the sensors connected to the PL OBC. Finally, $s(t)$ is the memory storage capacity inside the PL OBC. Some further explanation will be provided when the dynamics of the problem are presented.

To propagate correctly the modes of the system, it is also required to save the current parameters of the algorithm. For the implementation, several state variables are added that save the current parameters like output accuracy and CPU load utilization. The input variables $A_A^{act}(t)$, $R_A^{act}(t)$, and $Pr_A^{act}(t)$ are matrices of dimension $n\ by\ 1$ where n is the number of algorithms. More details are provided when the dynamics of the system are described.

As mentioned in the definition of MINLP, the problem also needs input variables called "U". They are the control of the system and by modifying their values, the behavior of the avionic is changed. In this implementation, the optimizer is modifying these variables to better fit the imputed constraint. It means the user has no direct control over the variation of these variables. A crucial point is that all these input variables are discrete since they represent the modes of the various elements. As for the state variables, they are each time-dependent. Furthermore, since each type of element contains multiple instances, all these variables are matrices. For example, there are multiple sensors that each have a specific mode. The list below is describing them. In the list, each variable is associated with an example. In this case, the example is directly linked to the second payload avionic architecture that will be presented later in the chapter. Nevertheless, these variables can be modified to accept any other configuration of payload architecture.

- δ_S : Sensor activation, matrix $n\ by\ n$ with binary value ($[0;1]$) and n is the number of sensors. It is a diagonal matrix where each element of the diagonal represents the status of the component. Example (3 sensors, the first and the third one are turned on and the second one is turned off.) :

$$\delta_S = diag(\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- M_S : Sensor modes, matrix $m\ by\ n$ with binary value ($[0;1]$), m the number of sensors

and n is the number of modes. Each line represents the modes of one component, and uniquely one element can be 1. The element (in the row) set to 1 defines the mode being active. Example (3 sensors: the first one is in mode 2, the second is in mode 1 and the third is in mode 3.) :

$$M_S = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- M_A : Algorithm modes, matrix $m\ by\ n$ with binary value ($[0;1]$), m the number of algorithms and n is the number of modes. Each line represents the modes of one component, and one element can be 1. Again, the element set to 1 defines the mode being active. Example (4 algorithms: the first and fourth ones are in mode 1. The second and the third one are in mode 2.) :

$$M_A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

- M_O : Payload On-Board Computer mode, matrix $1\ by\ n$ with binary value ($[0;1]$) and n is the number of modes. Exclusively one element can be 1 and it represents the mode of the PL OBC. Example (PL OBC is in mode 3) :

$$M_O = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

This representation of the input variables has the advantage of formulating an easier mathematical definition of the problem. Indeed, by implementing matrix multiplication and extracting the diagonal, it is straightforward to retrieve the current parameters of the various elements. The principal disadvantage remains the number of input variables needed for the control. For example, if each algorithm has 3 modes, it needs $3 * n$ input variables. The initial design was using exclusively one variable per element, however, it was causing issues in the implementation of the dynamics. To tackle this issue, the method used has been to implement all the control variables as binary. Even though it is not a practical aspect when reading the raw output of the optimizer, it enables the tool to converge on a solution. This aspect is reworked in the next chapter.

2.2 Parameters of the System

This subsection is dedicated to the various fixed parameters of the simulation. They represent the physical and abstract characteristics of each element of the system. As for the input variable, the examples given reflect the second payload avionic architecture. Nevertheless, the flexibility of the implementation allows for other configurations.

- Output frequency by algorithms F_A ($n\ by\ 1$ matrix. n is the number of algorithms). It stores the frequency at which the algorithms are being updated. Example for 4 algorithms:

$$F_A = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

- Accuracy level by algorithms depending on mode A_A ($m\ by\ n$ matrix. n is the number of algorithms, m the number of modes). This constant defines what is the gain in accuracy after each algorithm update. Example of 4 algorithms with 3 modes:

$$A_A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{bmatrix}$$

- Loss rate of valuable information α. This parameter evaluates how long it takes for the information on the target to be irrelevant. The assumption is made that any algorithm update (or accuracy update) is valid for a defined amount of time. In addition, the confidence in the accuracy variable decrease over time. As an example, the spacecraft can compute the position and velocity of the target at a defined time. With some propagation algorithms (which are not shown here), the spacecraft can estimate the position of the debris after some time even though no new observation has been done. In any case, this orbit propagation process is subject to incertitude on the measurement and thus the confidence in the new position decreases the longer it wits. The goal of this constant is to represent this behavior.

- Electrical energy available in the payload avionic E_{Tot}. It varies over time with $E_{Tot}(t) = E_{min} + E_{amp} * sin(2\pi * f * t)$. In this equation, E_{min} is the offset, E_{amp} represents the amplitude of the variation and f is the frequency. This representation could be modified, but it needs to stay time-dependent. This modeling is entirely arbitrary and could represent any flight phase of the mission at a high level of abstraction. The driving factor behind this specific implementation is the variation of energy because of solar panels' orientation. For example, the frequency can be linked to the day and night phase of a LEO orbit. It can also relate to a spinning spacecraft that has not the option to orient its solar panels. By decomposing this variable, it can be assumed that the constant part is linked to the batteries of the spacecraft. This representation has limitations such as the energy available at the spacecraft level is not the same as the one for the payload side. In addition, the argument related to the energy limitation can be argued over a better sizing of the batteries. These points are discussed at the end of the chapter.

- Electrical energy consumption of sensors depending on mode P_S ($m\ by\ n$ matrix. n is

the number of sensors, m the number of modes). Example of 3 sensors with 3 modes.

$$E_S = \begin{bmatrix} e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,1} & e_{3,2} & e_{3,3} \end{bmatrix}$$

- Electrical energy consumption of the PL OBC depending on mode P_O ($m\ by\ 1$ matrix. n is the number of modes). Example with 3 modes.

$$E_O{}^T = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix}$$

- Processing resources or CPU load usage by algorithms depending on mode R_A ($m\ by\ n$ matrix. n is the number of algorithms, m the number of modes). It represents how much resources are needed by each algorithm. Example of 4 algorithms with 3 modes.

$$R_A = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & r_{1,4} \\ r_{2,1} & r_{2,2} & r_{2,3} & r_{2,4} \\ r_{3,1} & r_{3,3} & r_{3,3} & r_{3,4} \end{bmatrix}$$

- Processing resources or CPU load available in the PL OBC depending on mode R_O ($m\ by\ 1$ matrix. n is the number of modes). This parameters shows what amount of CPU load the PL OBC is able to support. Example with 3 modes.

$$R_O{}^T = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}$$

- Data rate by sensors depending on mode D_S ($m\ by\ n$ matrix. n is the number of sensors, m the number of modes). Its shows ho much data is produced by each sensors. Example of 3 sensors with 3 modes.

$$D_S = \begin{bmatrix} d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,1} & d_{3,2} & d_{3,3} \end{bmatrix}$$

- Processing data rate by algorithms depending on mode Pr_A ($m\ by\ n$ matrix. n is the number of algorithms, m the number of modes). This parameter represents how much data is consumed by the algorithm to compute their outputs. Example of 3 algorithms with 3 modes.

$$Pr_A = \begin{bmatrix} pr_{1,1} & pr_{1,2} & pr_{1,3} & pr_{1,4} \\ pr_{2,1} & pr_{2,2} & pr_{2,3} & pr_{2,4} \\ pr_{3,1} & pr_{3,2} & pr_{3,3} & pr_{3,4} \end{bmatrix}$$

As it can be seen, all the parameters reflect a general high-level of abstraction of the systems. The sensors at this point do not represent any specific camera or instrument. In addition, the algorithms are very generic and have only high-level parameters. At this level of implementation, these items are "consuming" data produced by the sensors and outputting an accuracy

update at a defined frequency. In parallel, they need processing resources or CPU load to run which is provided by the PL OBC. The advantages and disadvantages of this implementation are discussed later. Regarding the behavior, more details are given about the dynamics of the system.

2.3 Constraints

The list below indicates the constraints applied to the state variables.

- Accuracy information on the target between 0 and 1. As specified before, this theoretical parameter is arbitrarily constrained on this small interval. A high value means high confidence about the computed output of the algorithms. All the values in-between the extremes represent a fraction of the "perfect" accuracy.

$$0 \leqslant c(t) \leqslant 1 \tag{4.1}$$

- Available electrical energy greater or equal to 0 $[W \cdot h]$. It represents the difference between what is being produced and consumed. Logically, the energy at the payload level should be above zero. An important point to mention is that no margin is taken here. Nevertheless, it is possible to change this lower limit to a higher value.

$$e(t) \geqslant 0 \tag{4.2}$$

- Available processing resources (CPU load) at the PL OBC greater or equal to 0 $[MIPS]$. This limitation is about the capability of the PL OBC to support algorithms. As for the previous constraint, the lower limit can be increased to include margin.

$$r(t) \geqslant 0 \tag{4.3}$$

- Data rate coming to the PL OBC greater or equal to 0 $[Mbit/s]$ and less than a maximum. The goal of the lower limit is physical. For the upper one, it is a design choice that has to be taken by the user of the tool depending on its requirement.

$$0 \leqslant d(t) \leqslant max_{data\ rate} \tag{4.4}$$

- Storage capacity greater or equal to 0 $[Mbit]$ and less than a maximum. As for the previous constraint, the lower limit is linked to the physical properties. The upper limit has to be fixed by the user.

$$0 \leqslant s(t) \leqslant max_{storage} \tag{4.5}$$

There are, furthermore, constraints on the input variables. Each of them is forced to be either 0 or 1. As specified before, the input variables are all binary and thus this constraint has to be written mathematically. For δ_S, it means the sensor is turned on or off. For the other input variables, it defines which mode is being used. Examples are shown at the beginning of the section.

$$U \in [0;1]$$

Except for δ_S, additional constraints need to be written in the system to satisfy physical properties. Logically, every element can have one and exclusively one mode active at the same time. It is physically not possible for a component to be operated in two different mode at the same time. Mathematically, it translates into all the matrices defined for the input variables should have each of their lines exactly equal to 1. The three equations below are for respectively the sensors (Equation (4.6)), the algorithms (Equation (4.7)) and the PL OBC mode (Equation (4.8)).

$$\sum_{j=1}^{n} M_{S_{i,j}} = 1, \ \forall i \in [0;m] \tag{4.6}$$

$$\sum_{j=1}^{n} M_{Ai,j} = 1, \ \forall i \in [0;m] \tag{4.7}$$

$$\sum_{i=1}^{n} M_{Oi} = 1 \tag{4.8}$$

2.4 Dynamics

This subsection is dedicated to the dynamics of the system. To simplify the various equations, three matrix operators are used. There is the $diag()$ that extracts the diagonal of a matrix and returns it in a vector form. $trace()$ provides the sum of all diagonal elements of a matrix. Finally, $sum()$ outputs the sum of all elements in a vector. These operators allow a straightforward definition of the system.

The three equations under (4.9) are written to extract the current parameters of the algorithms depending on which mode they are running. These three variables are needed for the update of the state variables A_A^{act}, R_A^{act}, and Pr_A^{act} as written in the equations (4.12), (4.13), and (4.14). The variables are the accuracy output, the CPU load usage, and the processing data rate of each algorithm. This step is important for the system since the matrices are updated at every iteration of the tool. It is fundamental to remember each of these variables is a matrix of size

$m\ by\ 1$ where m is the number of algorithms.

$$\begin{aligned} A_A^{new}(t) &= diag(M_A(t) * A_A) \\ R_A^{new}(t) &= diag(M_A(t) * R_A) \\ Pr_A^{new}(t) &= diag(M_A(t) * Pr_A) \end{aligned} \tag{4.9}$$

Equations (4.11), (4.12),(4.13), and (4.14) are applied for every algorithm with the definition given in (4.10).

The equation (4.11) simply expresses if an update has been done by one of the algorithms. $up_i\,t$ is set to 1 if it is the case. The logic is to take the modulo of the period of each running algorithm and compare them with the actual and previous timestamps. If the modulo of the previous timestamp is greater than the actual one, then there is an update. In this case, the output of the algorithm that received its update is saved. To be noted that this operation is done for every algorithm of the system as specified in equation (4.10). These two equations are defining at what frequencies the accuracy parameter should be updated. Importantly, the algorithms do not have to be in the same phase nor have the same contribution to the accuracy.

$$\forall i \in [0; m] \tag{4.10}$$

$$up_i(t) = \begin{cases} A_i^{act}(t), & \text{if } (t - dt)\ mod\ \dfrac{1}{F_{Ai}(t)} > t\ mod\ \dfrac{1}{F_{Ai}(t)} \\ 0, & \text{otherwise} \end{cases} \tag{4.11}$$

The following step is to determine if the parameters of the algorithm need to be updated with the equations (4.12), (4.13), and (4.14). The variables updated are the accuracy output, the CPU load usage, and the processing data rate of each algorithm. The principle is to check if the variable $up_i(t)$ is greater than 0 for every algorithm and update the state variables when it is the case. If no update is required, then the derivative is just set to 0 to maintain the same value for the subsequent iteration. In the other case, the variable derivative takes the differences between the new value and the previous one divided by the time step. This allows these three variables to store the change of parameters inside each algorithm. Nevertheless, this derivative approach has multiple drawbacks that would be discussed later.

$$\dot{A_A^{act}}_i = \begin{cases} \dfrac{A_A^{new}{}_i(t) - A_A^{act}{}_i(t)}{dt}, & \text{if } up_i(t) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.12}$$

$$\dot{R_A^{act}}_i = \begin{cases} \frac{R_A^{new}{}_i(t) - R_A^{act}{}_i(t)}{dt}, & \text{if } up_i(t) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.13}$$

$$\dot{Pr_A^{act}}_i = \begin{cases} \frac{Pr_A^{new}{}_i(t) - Pr_A^{act}{}_i(t)}{dt}, & \text{if } up_i(t) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.14}$$

The dynamics of the main state variables are presented below. Every time, the derivative of the state variable is shown.

Equation (4.15) represents the decrease of accuracy on the target over time. At a given point, some information about the target is known. Then the more time goes by, the less this information is relevant. This loss is described with the α parameter that sets the derivative to a negative value. It means the accuracy parameter decrease linearly when no update is happening. In the case of an update, the (4.16) provides the highest accuracy produced by the algorithm and then uses it to compute the slope of the curve.

$$\dot{c}(t) = \begin{cases} \frac{up_{max}(t) - c(t)}{dt}, & \text{if } up_{max}(t) > 0 \\ -\alpha, & \text{otherwise} \end{cases} \tag{4.15}$$

$$up_{max}(t) = max(up_i(t)),\ \forall i \in [0; m] \tag{4.16}$$

Equation (4.17) is the electrical energy available. The usage of the PL OBC and the sensors are subtracted from the energy produced by the spacecraft. For the sensors, δ_S is used to determine if the component is turned on and then the mode defined how much energy is consumed. Regrading the PL OBC, the mode matrix is multiplied by the energy consumption matrix for similar reasons. This result is compared to the effective electrical energy remaining and then divided by dt to obtain the derivative.

$$\dot{e}(t) = \frac{E_{Tot}(t) - trace(\,\delta_S(t) * M_S * E_S\,) - M_O(t) * E_O - e(t)}{dt} \tag{4.17}$$

Equation (4.18) is the processing resources or CPU load left. It follows an identical principle as before. The mode matrix of the PL OBC is multiplied by the processing resources output matrix and then the consumption of all the algorithms is subtracted. Since R_A^{act} represents already a vector with the current algorithm processing resource usage, a sum of all elements is needed. With this result, the derivative can be computed again. In other terms, this equation

shows the balance of CPU load between what is available and what is consumed.

$$\dot{r}(t) = \frac{M_O(t) * R_O - sum(R_A^{act}) - r(t)}{dt} \tag{4.18}$$

Equation (4.19) represents the sum of all data rates produced by the sensors and going into the PL OBC. To be remembered that the state of the sensor is important in this case since it can have no data production if it is turned off. The same principle as before is used to obtain the derivative.

$$\dot{d}(t) = \frac{trace(\ \delta_S(t) * M_S * D_S\ ) - d(t)}{dt} \tag{4.19}$$

Equation (4.20) is the storage capacity used by the sensors and algorithms. In this case, the inputs of the sensors are taken and then the algorithms' consumption is subtracted. It is an equilibrium between what is produced and what is consumed. This specific behavior is discussed later.

$$\dot{s}(t) = \frac{sum(\ \delta_S(t) * M_S * D_S\ ) - sum(M_A(t) * Pr_A)}{dt} \tag{4.20}$$

2.5 Cost/Objective Function

The last essential point about the problem formulation is the objective function. It is the core of the simulation since it indicates which parameters are fundamental for the optimizer and to which level.

The cost function (4.21) possesses three main elements that need to be minimized over the whole time interval. As a reminder, K constitutes the number of steps done inside the optimizer, and for each step, a defined duration is attributed. The function needs to be the smallest overall but not specifically at every time step. Moreover, each of these conditions includes a parameter (β, γ, or λ) that needs to be tuned.

$$min \sum_{i=1}^{K} \beta * \frac{nb\ sensor - sum(\delta_S(t_i))}{nb\ sensor} + \gamma * (1 - c(t_i)) + \lambda * \frac{s(t_i)}{max_{storage}} \tag{4.21}$$

The first item in the function $\beta * \frac{nb\ sensor - sum(\delta_S(t_i))}{nb\ sensor}$ is related to the sensor being turned on. It expresses the need for the maximum number of components working simultaneously. The system wants to obtain the highest amount of information coming out of all sensors, so the function forces it to have most of them turned on. Since information is key for the payload avionic, it is crucial to obtain the largest amount of data.

The second condition $\gamma * (1 - c(t_i))$ is primarily on the highest desired accuracy on the target. The closer it is to 1, the more confident the system is about the shape and the motion of the target. It is one of the essential factors of the cost function since mission success is primarily dependent on this information. Arguably this requirement can change depending on the phase of the mission, nevertheless, it stays a decisive parameter.

The third one $\lambda * \frac{s(t_i)}{max_{storage}}$ is here to minimize the amount of data that stays in the memory. In relation, the algorithms have to consume the majority of information produced by the sensors. To the same level, the sensors must output a large amount of data, it is capital for the algorithm to employ all the information produced for their analyses. The function works with unitless parameters, it is why a percentage of memory used is given. As for the memory itself, this implementation has some limitations that are discussed in the next subsection.

Regarding the parameters of the cost function (β, γ, or λ), they have to be selected carefully to emulate the desired behavior. As stated before, these three variables define the importance of each element of the function. During the design process, many iterations have been required to obtain meaningful results. Moreover, it frequently made the problem infeasible for the optimizer. The key was to analyze the result of the optimizer and to analyze if the system behavior matched the expected output. Ultimately, it was selected arbitrarily that each one should be between 0 and 1. For the set of simulations presented later, it has been decided to retain the second term as the most significant one followed by the first one and the third one.

2.6 Challenges

One of the challenges of this MINLP model is its accuracy. Since the representation implemented is at a high-level of abstraction, it is arduous to validate such a model. The decisive variable is, of course, the accuracy on the target which has been defined as an indicator of the precision of the algorithms. It is untrivial to define such parameters and have them linked to physical properties without extensive analyses. At this stage, the various algorithms that will be used by CS-1 are still in development. Meaning it is barely possible to have a first evaluation of their precision and accuracy. In addition, the goal of the tool is to stay at a general level and to apply to various types of missions. Another factor that is not taken into consideration is the variation of this accuracy depending on the mission phase and the distance to the target. The decision to stay with this highly abstract parameter has been made to simplify to development of the tool and enable faster development. Nevertheless, this decision is discussed at the end of the chapter.

Linked to the accuracy parameter is the general level of abstraction from the tool and the model. In one direction, it is mostly straightforward to update the models of the various elements with more representative information. It would definitely decrease the incertitude on various parts of the optimization. Nevertheless, the tool itself has some general abstraction implemented in its structure. It is an aspect that would need the code of the tool to be modified which is a level of complexity higher.

Few comments regarding the general implementation are also required. The first one is regarding the memory inside the payload avionic. At this stage, this element is highly generic and simply stores all the information provided by the sensors. On the other end, it deletes information as soon as it is used by the algorithms. In a real architecture, it is doubtful that all information from the sensors would be stored. The most probable solution would be to store the important information and forward to the algorithm the needed part. If some information is not needed and important, then the data is simply not stored or kept. This behavior is not included in the optimizer.

A topic already talked about is the usage of electrical energy at the payload level. This decision is not ideal since the energy usage highly differs from component to component. For example, an electric propulsion system would need several orders of magnitude more energy than a simple camera. It means turning off a camera in a phase of the flight would have almost no impact at the system level if the propulsion is working. It means that energy constraint at the payload level lacks the overall view of the system. It also means that driving the system by using this constraint might not be representative.

The final critical challenge is to determine if the solution provided is a global optimum. Because of the complexity of the system and the various constraints applied to it, the output produced by the optimizer might be a local optimum. More rigorous analyses will be carried out to address this issue, especially regarding the impact of the various parameters. In addition to this point, it is crucial to assess the implementation of the derivative for the system dynamics. As it will be shown later, the system's behavior is not fully representative of an actual scenario. This effect is probably linked to the dynamics' implementation. More information is provided in the comments part of the chapter.

3 Simulation results

In this section, two scenarios are presented with various sets of parameters. The goal is to generate a preliminary idea of the system's behavior and evaluate the capability of the optimizer. Exclusively two scenarios are selected to emphasize the comparison between them and provide some preliminary assessment of the simulator. As stated before, the payload avionic architectures are highly abstract to test the basis of the optimizer. The subsequent iteration will allow going deeper into the complexity of the architectures and output more valuable insights.

3.1 Mission Architecture

The two unique payload architectures (Figures 4.1 & 4.2) have been tested with some variation in their parameters. Both of them are relatively similar in their implementation with a notable difference in the number of elements they contain. The second architecture represents a more complex configuration than the first one. In both cases, they contain very generic sensors

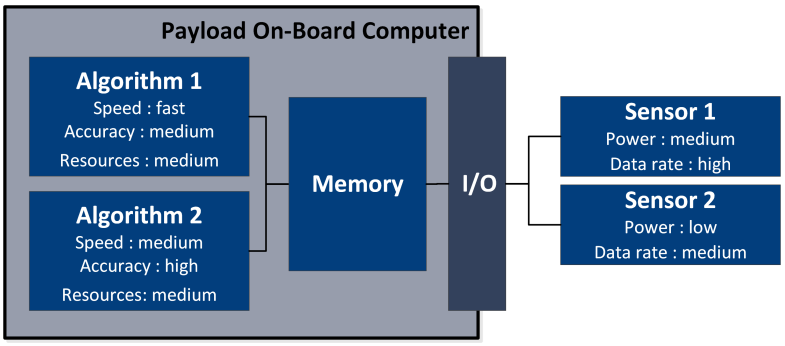


Figure 4.1 – **Payload avionic architecture 1.**

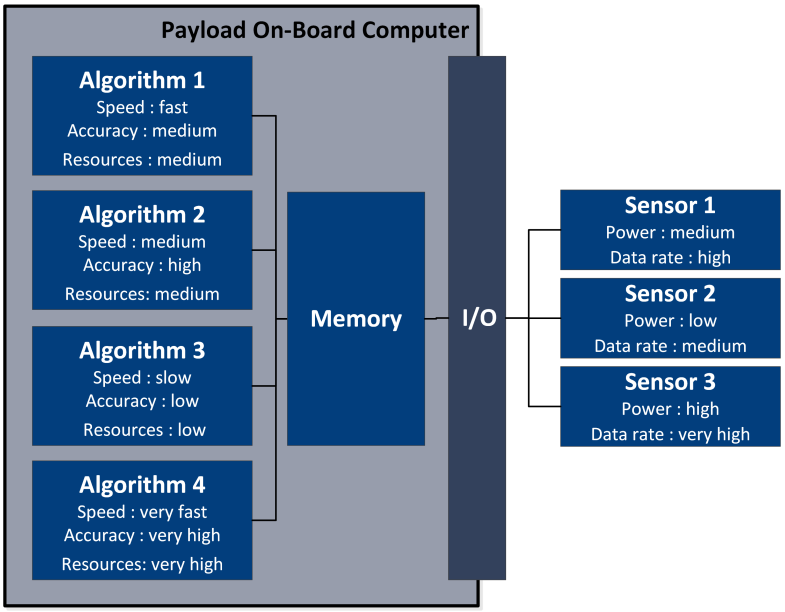


Figure 4.2 – **Payload avionic architecture 2.**

and algorithms with some performance differences. As explained earlier, the goal is to test the performance behavior and not a specific realistic payload architecture. In addition, these payload architectures are slightly different from the previous chapter to better fit the tool capability.

The first payload architecture is presented in Figure 4.1. This avionic includes two sensors connected to the PL OBC with both of them supporting three unique modes. The first mode is typically called LE since it has the lowest energy consumption among the three. It represents a sleeping state where the element is just being kept alive and providing few data. The two other modes are here to reflect various levels of usage. The third one is usually more powerful than the second one. For the sensors, it means they have a higher data rate but consume more electrical energy. Regarding the two components, no specific instruments have been chosen. They represent some generic types of sensors. In comparison to each other, the first one is

slightly more powerful than the second one but it also consumes more electrical energy.

The architecture additionally includes two algorithms that both run on the PL OBC. As for the sensor, these elements include three distinctive modes with the first being the LE one. Regarding their parameters, both algorithms run at different frequencies which do not depend on their states. In opposition, the output accuracy, the processing resources usage, and the data rate consumed are all mode dependent. The modes are similar to the sensor with the second one being less powerful than the third one. In comparison to each other, the first algorithm is less accurate. Nevertheless, it consumes fewer processing resources (or CPU load) of the PL OBC, requires fewer data produced by the sensors, and has a higher frequency. As for the sensor, the algorithms are very generic and are not linked to any actual element.

The last element of the architecture is, of course, the PL OBC. As for all the other elements, it includes three distinctive modes and the first one is LE. The third mode is again slightly more powerful than the second one but consumes more electrical energy.

The second payload architecture is presented in Figure 4.2. As written beforehand, this payload avionic represents an upgrade of the first one. The characteristics of the PL OBC, the two first sensors, and the two first algorithms are identical. In addition to the first avionic, it includes one additional sensor and two newer algorithms. Nevertheless, all the elements contain three modes with the first one invariably being LE.

Regarding the extra sensor, it has a significant improvement in terms of output data rate, however, the element also consumes a larger amount of electrical energy.

For the additional algorithms, the first one remains a more classic version of the first two. It has lower accuracy and frequency but the entity consumes less processing resources (or CPU load). In opposition, the fourth algorithm is more complex than the previous ones. This element provides the highest output accuracy and frequency but it requires the largest amount of processing resources.

3.2 Solutions

The optimizer has been used with the two payload architectures presented in the previous subsection. In addition, two scenarios have been applied to the first architecture to make a further comparison of their behavior. For both cases, the scenarios are very generic and can reflect a final approach of a debris. The goal was not to define a very specific set of constraints but to apply a generic change to the payload avionic.

Regarding the first scenario, it was decided to put some constraints on the system through the electrical energy available in the satellite. In this case, a minimum amount of electrical energy is available and then there is the variation due to the solar panel orientation. This can be seen as if the satellite has its batteries recharged and then enter an operation mode. The operation could imply some rotation of the satellite with a non-optimal orientation of the solar panels. It

would mean some variation in the available energy. The optimizer has to make a compromise between the mode of the PL OBC and sensors as well as the number of sensors. Nevertheless, it is a fictive operation mode that has been created to apply constraints to the payload avionic. In this case, the flight phase is very short and has a duration of 2 $[min]$.

The second scenario is almost the same as the first one with the notable exception of the frequency of the electrical energy production. This fictive situation would mean that the satellite is rotating faster. To compare the effect of the energy on the system, the supply produces the corresponding level of energy but with a frequency that is twice the previous one. The duration of this second scenario is identical to the first one. This unique change was deliberate to compare the payload architectures between them and one architecture with two slight differences in the constraints.

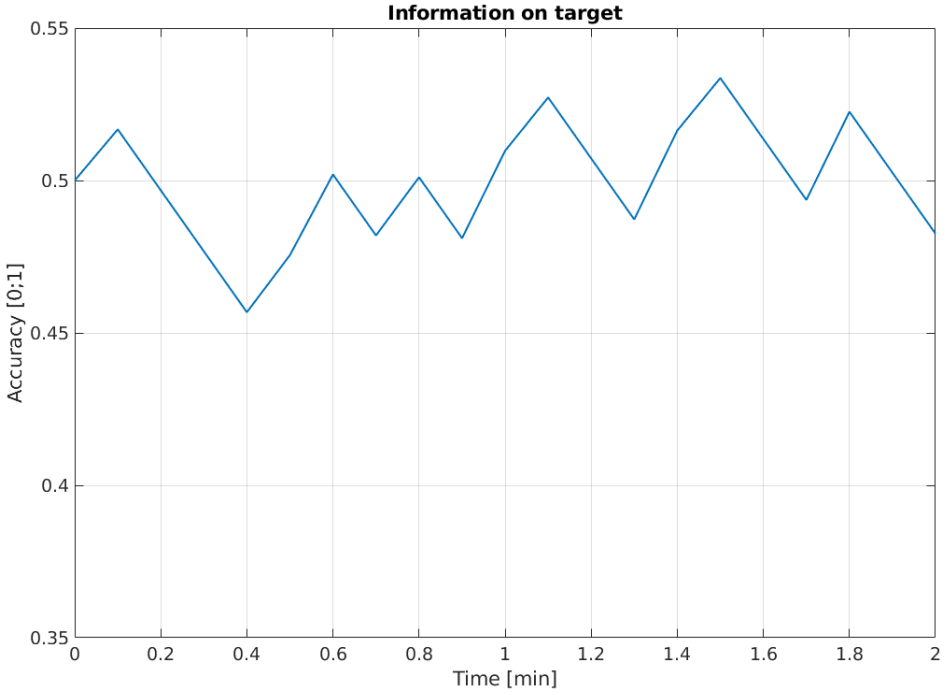


Figure 4.3 – **Payload avionic 1 - Scenario 1 - Accuracy on target.**

The Figures 4.3, 4.4, and 4.5 show the results of the first optimizer test with the first payload avionic architecture (Figure 4.1) and first scenario. In this case, the accuracy on the target stays pretty stable around the value of 0.5. It has some oscillation due to the frequencies of the algorithms but overall the value of accuracy is not decreasing significantly. Relatively to the other state variables, the energy consumption is not a constraint in itself since it maintains a minimum of 100 $[W \cdot h]$ all the time. In comparison, the processing resources available drop rapidly and then stay close to 0 $[MIPS]$. It can be noted that the slight drops in the processing resources are directly correlated with the spike in electrical energy consumption. This effect is linked to Figure 4.5 and the modes' variation of the PL OBC. With the new mode, the energy

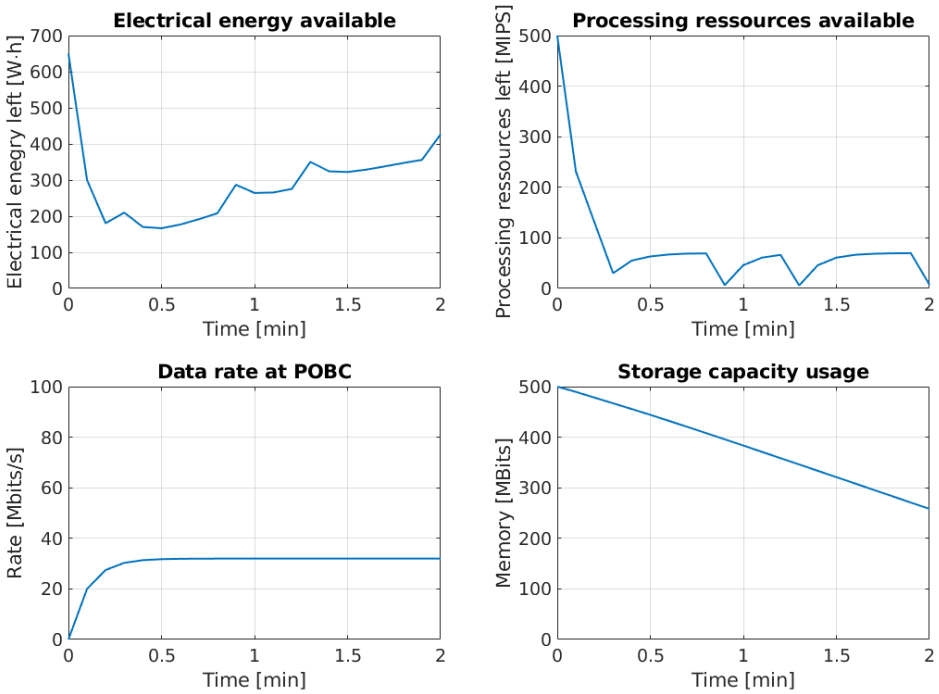


Figure 4.4 – **Payload avionic 1 - Scenario 1 - State variables.**

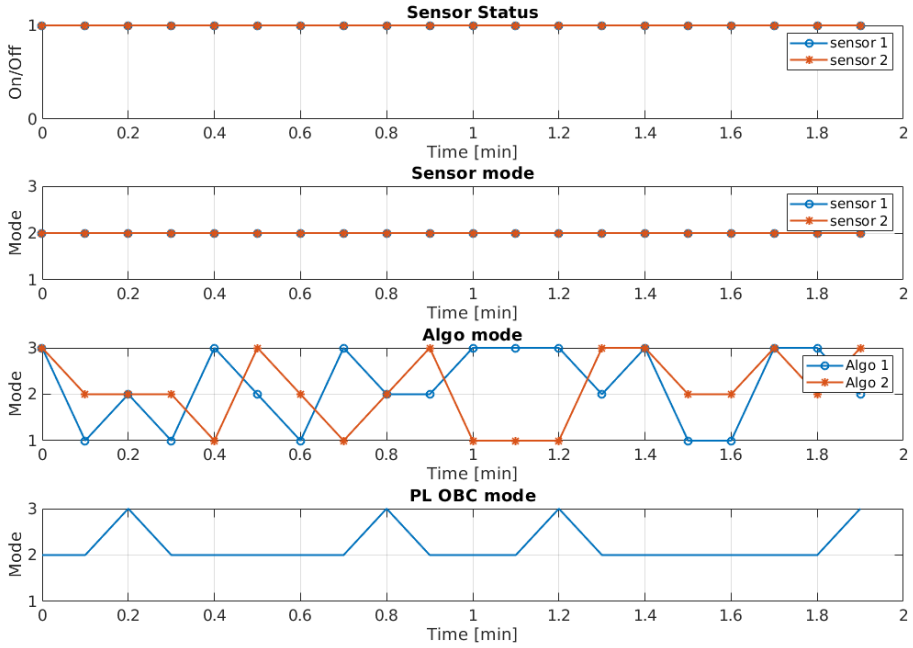


Figure 4.5 – **Payload avionic 1 - Scenario 1 - Input variables.**

available is increasing (since it consumes less) but the processing resources are decreasing.

Regarding the input variables in Figure 4.5, both sensors stay on during the whole simulation. It is a direct effect of the objective function which penalizes the system for not keeping them turned on. Surprisingly, the sensors' states are not varying over time. The optimizer sets both of them to mode 2 and keeps it for the full duration. There are more activities in the algorithms and PL OBC mode. Unusual behaviors can be observed in the control of the algorithms. The optimizer is almost all the time altering the mode of these elements and it is barely using the first one. It can be remarked that the two algorithms are repeatedly not in the same mode. One solution for this high oscillation could be the constraint due to the processing resources. As it is seen in Figure 4.4, the available resources are often close to 0 $[MIPS]$. This behavior could influence the optimizer to alter frequently the mode of the algorithms to keep the accuracy to the same level. The algorithm behavior is obviously something that needs to be tackled. It is not acceptable from a mission planning point of view to have such changes. The relevance of such scenarios is discussed later.

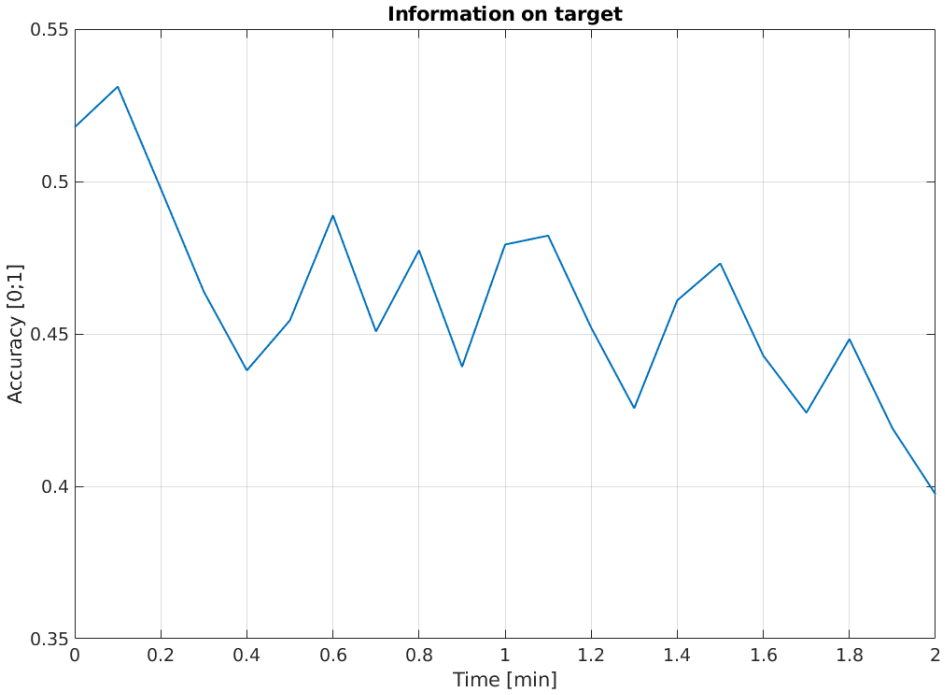


Figure 4.6 – **Payload avionic 1 - Scenario 2 - Accuracy on target.**

With the same avionic as before and the higher frequency of the electrical energy supply (scenario 2), it can be observed similar behavior as previously for most state variables. One exception remains the accuracy on the target which shows an overall decrease in Figure 4.6. The decrease can not be linked to the overall higher constraint regarding the electrical energy and has to come from another variation in the system. Since the energy available seems to be higher in the second case than in the first one, this result may have run into the problem of

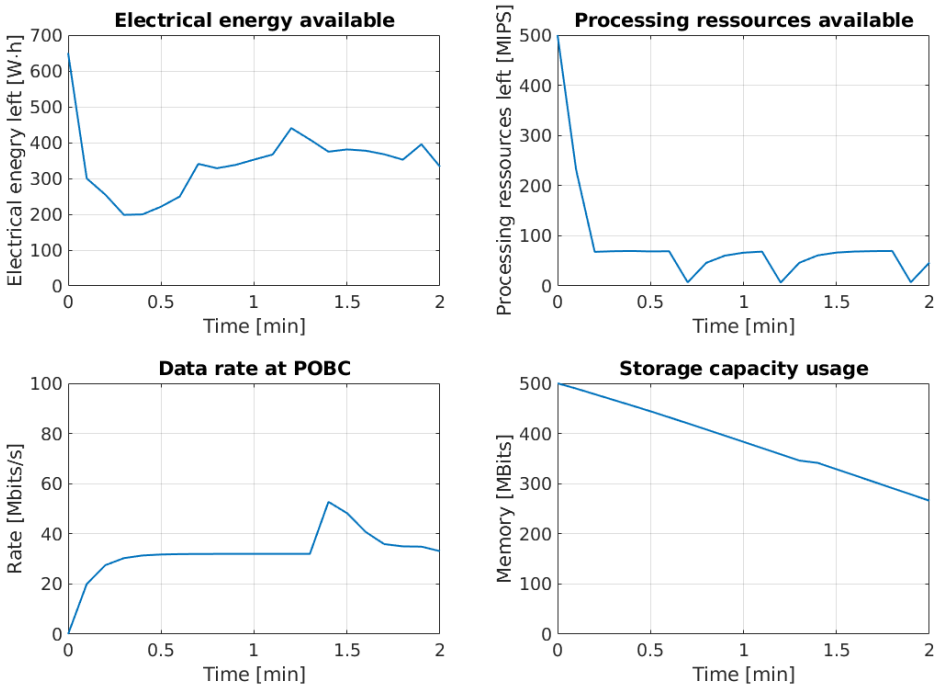


Figure 4.7 – **Payload avionic 1 - Scenario 2 - State variables.**

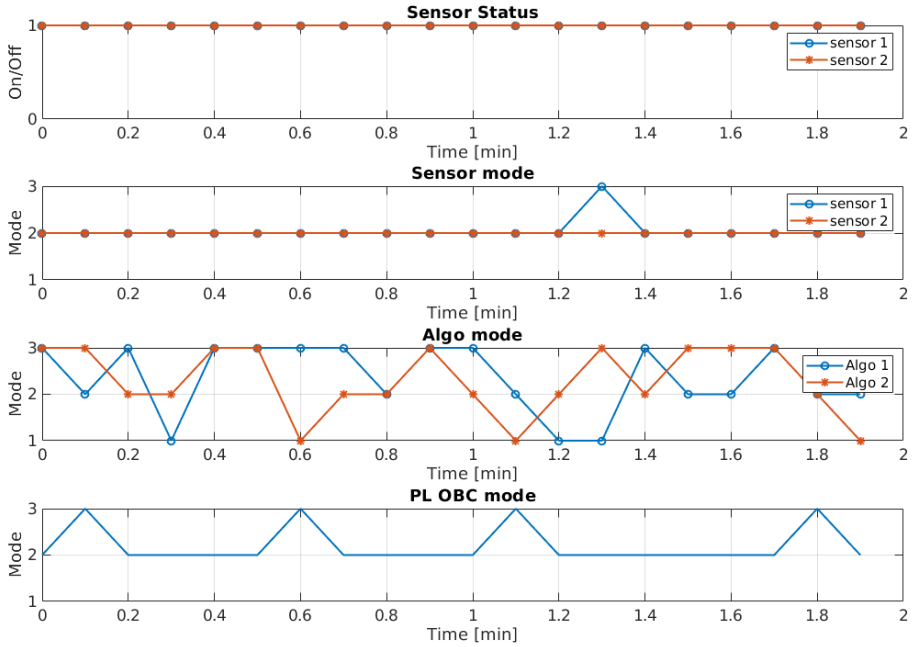


Figure 4.8 – **Payload avionic 1 - Scenario 2 - Input variables.**

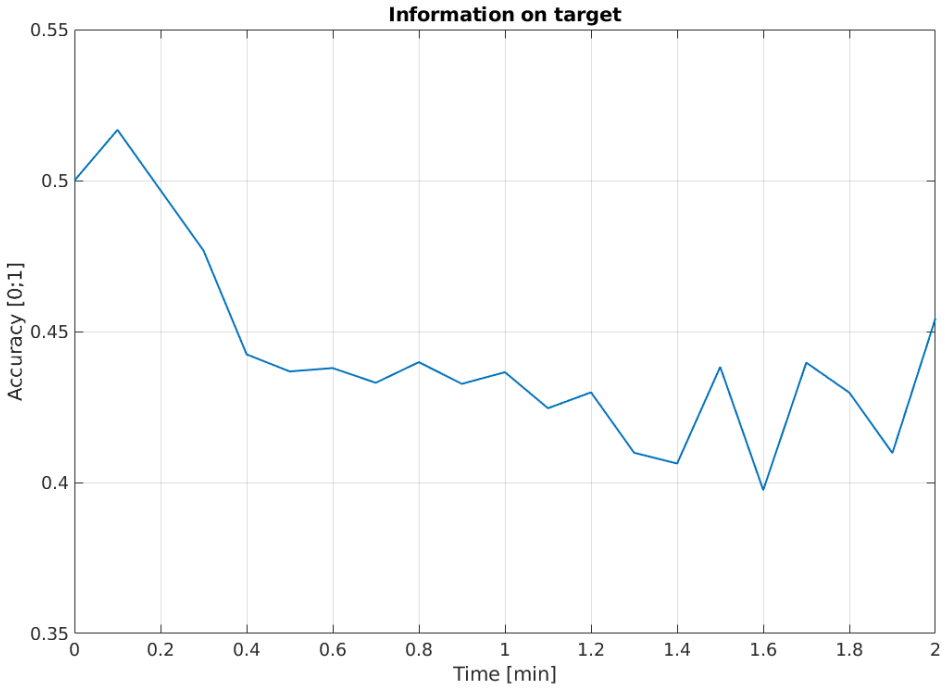


Figure 4.9 – **Payload avionic 2 - Scenario 1 - Accuracy on target.**

a local maximum. Indeed, because of the similar level of energy, the result with the second scenario should be practically identical.

Regarding the other state variables in Figure 4.7, all of them seem similar to the previous scenario with obviously the exception of the energy available. The peak of the supply can be observed in the second case, and it does not look as if the system is employing more energy than previously. The CPU load are again following a similar behavior as before with some drops close to 0 $[MIPS]$. The drops are directly linked to the mode of the PL OBC. A spike in the data rate at the PL OBC can also be observed in the bottom left graphic. Its cause can be pinpointed in the mode of sensor number 1.

The Figure 4.8 shows the variation in states of the avionic. As for the first scenario, the sensors are habitually turned on and stay almost exclusively in mode 2. The algorithms are additionally undergoing some high oscillation in their mode variation. The reason behind these rapid changes can be linked to the constraint on the processing resources available.

The Figures 4.9, 4.10, and 4.11 are the results for the second payload avionic (Figure 4.1) with the first scenario. The key difference with the other architecture remains the shape of the accuracy on target. The reason comes from the increase in the number of algorithms running. Since more of them are being used by the system, there is overall more update. Nevertheless, the accuracy on target is dropping to a 0.42 level. It is problematic to predict the reaction of the system after these two minutes, but it could stabilize. In any case, this behavior might be

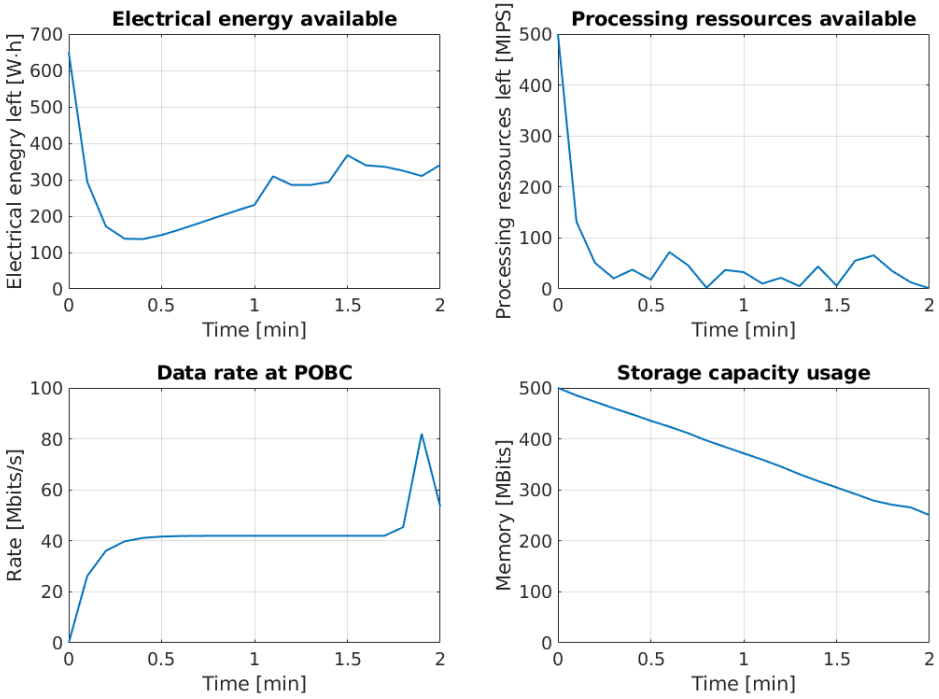


Figure 4.10 – **Payload avionic 2 - Scenario 1 - State variables.**

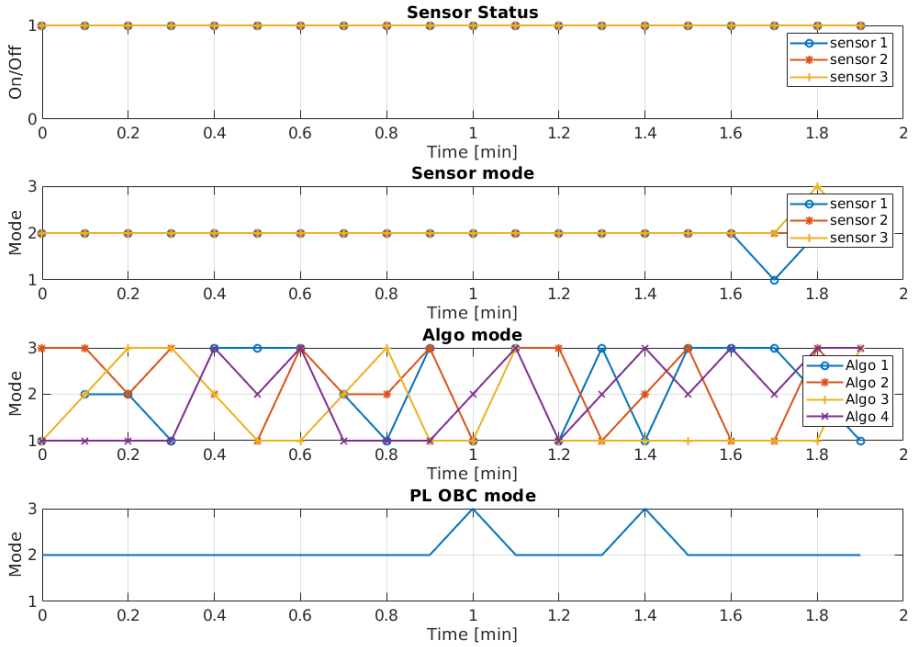


Figure 4.11 – **Payload avionic 2 - Scenario 1 - Input variables.**

indicating another underlying issue. Since this architecture has more sensors and algorithms than the first one, it should output a better result.

The other state variables follow roughly a similar trend as in the first avionic. The electrical energy available rapidly rises to a 300 $[W \cdot h]$ level, and it does not affect the system significantly. In opposition, the processing resources drop to almost 0 $[MIPS]$ and stay at this level during the whole simulation. Another unusual event remains the peak in the data rate that occurs almost at the end of the simulation. Its origin is directly linked to the change of mode in the sensors.

Regarding Figure 4.11, similar behavior as for the previous simulations can be observed. The three sensors stay on the whole time and their modes are almost entirely fixed to the second one. The PL OBC equally undergoes a few changes in modes that seem to barely impact the behavior of the system. Ultimately, there are once more high oscillations for the algorithms. It is assumed that the limitation in the processing resources is at the origin of these fluctuations.

3.3 Comments

Some general comments can be extracted from the three optimization runs. The first major observation remains the impact of the first term of the objective function (Equation (4.21)). To be noticed that the first term enforces the use of the maximum number of sensors. Its effect can be directly perceived as all the sensors are always turned on during the whole simulation. Ultimately, it acts as if the input variable δ_S were useless. If they were not implemented in the system, the result would be identical. Two options can be envisioned at this point. The first one would be to decrease the weight of the component in the cost function, which means having a smaller β. By doing so, the optimizer would be able to relax the constraint on the condition, and peculiar behavior could appear. The second option is linked to another general comment and concerns the electrical energy. One reason for the sensor to be always turned on is their low energy consumption and the slight impact they have on the overall electrical state of the system. For the optimizer, it makes almost no difference to maintain all the sensors turn on since they have a minor consumption.

The second major comment for the system is about the electrical energy supply in the avionic. In the three simulations presented previously, the energy production was almost always enough for the system to operate at maximum efficiency. As it has been seen in the previous results, the variations between the first and the second scenario were marginal. To perceive the concrete effect of this component on the whole system, more simulation would have to be done with much less electrical energy available. It will be the topic of the following chapter. On the same topic, it would be important to include energy constraints from the whole spacecraft and not only the payload avionic. As stated earlier, the energy consumption of the sensor can be marginal in comparison to other spacecraft subsystems.

Regarding the behavior of the algorithms, some modifications are needed. As seen in all cases

tested here, the optimizer has the ability to modify the algorithm mode at any time. It is not something that is realistic and some constraints are implemented in the next version. Overall, the stability of the system needs to be enhanced.

Another important point is the behavior of the accuracy on target. The three tests have shown some fast and almost unpredictable modifications of the parameter. The increase of the parameter from an algorithm has the same impact as the time decrease. It is not a realistic result even though the parameter is abstract. This effect is probably linked to the design issue regarding the impact of the parameter on this output. It is a behavior that is partially addressed in the following chapter.

By extension from the previous comment, some arguments can be made against the current implementation of the dynamics. The model tends to overreact to some changes as it can be seen with the various spike in the figures. In addition, the reaction of the system tends to be faster than expected resulting in unusual variation. The choice to use derivatives for the dynamics is a design choice made regarding the tool used. Nevertheless, it shows its limitations in this application and the need to modify the optimizer implementation. This particular aspect leads to the design of the next optimizer iteration. Another reason for this modification is the difficulty of the tool to converge to a solution. Even though the number of optimization steps was kept to a low level and the payload architecture to a low complexity, the tool required several hours to converge.

The final comment about this first test phase is about the scenarios. As explained above, the scenarios do not reflect any real flight phase of an ADR mission. The goal was only to test the sensitivity and look at the behavior of the tool. Nevertheless, it would be highly advantageous to move to a more realistic basis. It is even more critical to be able to analyze the result correctly. Indeed, if a realistic flight phase is emulated, then the optimizer results can be compared to plannings from the literature review.

4 Outcome & Incoming Challenges

The optimizer has demonstrated some considerable potential in defining the optimal payload avionic architecture. The ability to modify the dynamics and the constraints on the system enables substantial flexibility in following the evolution of the payload avionic platform for CS-1.

The three simulations presented in this chapter have proved the capability of the tool for more complex architecture analysis and optimization. The ability to compare two unique architectures with the same set of constraints is a fundamental advantage when designing the final avionic. Moreover, the output of the simulator in terms of control is a essential indication of the feasibility of the avionic. It is more straightforward from a mission planning point of view to start with an existing optimal solution in other to extract a feasible strategy. Furthermore, the usage of the sensors and algorithms is valuable information regarding their

necessity in the final architecture. In addition, the general approach of the tool considers the optimization of components usage over a defined period of time. A different method could consist of optimizing the task allocation. This topic is discussed at the end of the thesis.

This tool is obviously unfinished and among the various improvements that still need to be done, there is the issue with the input variables. Indeed, the optimizer requires one variable per mode and per element, which is extremely inefficient. Moreover, this implementation means additional constraints need to be set for the system to work properly. A solution would be to store directly the mode as an integer and then transform it into the dynamics part of the problem. This implementation was tested unsuccessfully because of the transformation of one input variable into a matrix which was unsupported by the optimizer. Other workarounds exist and should be investigated. Ultimately, by decreasing the number of input variables, the general system would be quicker to solve the problem. Even though the topic was unaddressed in this work, it is still an attractive option for eventual further development.

As mentioned in an earlier subsection, the accuracy of the model remains an issue that needs to be addressed. It is right now difficult to validate any outputs of the model since the level of abstraction is relatively high. Naturally, there are possibilities to validate part of it with a more simple avionic. Indeed, by creating a basic hardware setup, it is theoretically possible to compare part of the result with the optimizer output. Evidently, the tool is designed for initial assessments of avionic architectures and thus it makes the validation less decisive. Nevertheless, flexibility needs to be incorporated into the general structure of the optimizer to follow the development of the satellite. If the design choice and the properties of the instruments and algorithms can be integrated as they are defined for the mission, the tool will provide constant feedback on the general architecture. This specific issue will be the subject of a following chapter.

As explained in the chapter, some issues exist regarding the tool and its behavior. The dynamics implementation is a key problem that will be revisited in the next chapter. The three simulations have shown some limitations in the exactitude of the system behavior. In addition, the energy is considered as a subsystem level which makes this constraint more restrictive than probably needed.

Lastly, a crucial point that needs to be improved in this simulator is decreasing the generally high level of abstraction. The concerns are touching two major points which are respectively the data management and the inputs/outputs of the algorithms. Regarding the first one, it is naively assumed that all the data produced by the sensor are similar and no distinction is done when storing it. Visibly, a RADAR or a camera are not producing an identical type of information. For the second concern, it is again assumed that every algorithm can consume any data stored in the memory which is practically incorrect. Moreover, their outputs are currently providing a simple indication of the accuracy and are not considering the various type of information that could be produced. The avionic has, for example, two or three algorithms that output consistent type of information. As result, the optimizer would have to

select the most relevant one depending on the situation. On the same level, two algorithms can produce two unique types of information equally meaningful for the satellite. These changes are incorporated in the following version of the optimizer.

5 Acknowledgments

The authors would like to thank Prof. Dr.-Ing. Timm Faulwasser from TU Dortmund for providing resources regarding MINLP and feedback on the optimizer.

5 Parametric analyses

1 Introduction

This chapter is based on the publication "Optimal Control Approach for Dedicated On-Board Computer in Active Debris Removal Mission" presented virtually at the 35th Proceedings of the AIAA/USU Conference on Small Satellites 2021[154]. The content has been reviewed to match the structure of the thesis. The point of this chapter is to describe the evolution of the optimizer and its new ability to perform parametric analyses. The driving aspect is to be able to test various scenarios in a systematic way and draw conclusions on the effect of the model's parameters. For this purpose, a simple payload avionic architecture is employed with some slight modifications regarding the one of the previous chapter.

To considerably increase the representativeness of the simulation tools, a focus on the improvement of the mathematical model has been achieved. This novel approach addresses various issues encountered during the previous versions. It can better simulate the flow of information through the payload architecture and considers multiple types of data. In the current model, the simulator can operate algorithms at various speeds and employs specific links between them and the sensors with some memories. This chapter also introduces a broader variety of generic scenarios. Emphasis has been made on comparing one architecture with multiple scenarios. A parametric analysis approach has been employed to better comprehend the importance of each parameter. The goal is toward testing the sensitivity of the tool to some parameters and not to test accurate scenarios. This analysis allows to better understand the behavior of the tool and to determine which parameters drive the optimal solution. In a long term, it helps focus on specific aspects of the payload avionic architecture to obtain better performances.

The optimizer version maintains a remarkably similar purpose to the previous one. Ultimately, its goal is still to find the most appropriate set of sensors and algorithms to fit the constraint applied to the system. Its key differences are the implementation used and the amelioration it brought. The mathematical model benefits from a complete rewriting to better accommodate the requirement of the solver used. Some variables are removed, and many are included to

generate a more resilient model and less prone to infeasible situations. In addition, the ability to have frequency changes in the algorithm is made possible as well as linking multiple ones to the same output. Memories are additionally included in the model to make the interfaces between the algorithms and the sensors. Moreover, they accept additional constraints like storage and bandwidth limitation.

This optimizer implementation moved away from the Mixed-Integer NonLinear Program (MINLP) and has been written with a Mixed-Integer Quadratically Constrained Quadratic Program (MIQCQP) approach. As explained in the introduction, these two approaches have a few differences in the way the mathematical part is written. Nonetheless, minor adaptations have been performed on the MIQCQP to accept time-dependent optimization which results in an increase in complexity. Indeed, the model needs to have all its variables defined for every timestep with constraints linking them. The following subsection explains various details of the implementation. The code of the optimizer can be found in Appendix C.

2 Mathematical Model

In this section, the mathematical model is presented and explained. The tool employed to resolve the MIQCQP problem is the Matlab toolbox Gurobi [168]. This solver is dedicated to solving a broad range of MINLP and is known for his highly optimized implementation.

The goal of the mathematical model is to represent an abstraction of payload avionic architectures. The first application is the description of a future payload architecture for the CS-1 satellite, but the idea can be extended to a vast range of other missions. The fundamental point is to optimize the conception of high-performance payload avionic under tight constraints. This model contains a Payload On-Board Computer (PL OBC), multiple algorithms, some sensors, and a few memories. Almost each of these elements contains multiple modes that the optimizer can change to find the ideal configuration. The optimal solution is to obtain the highest accuracy on the target over time without breaking any of the constraints.

For the practical implementation of the optimizer, the model includes a list of variables that are all defined for every timestep k of the optimization. Some of them are completely handled by the optimizer, and others have constraints that need to be respected. The whole model can be written in form of linear and quadratic constraints with the addition of the variables' bound. The following subsections describe the reasoning and the implementation of the model.

In all these simulations, the most interesting variable remains the generic accuracy on the target. As it will be demonstrated in the objective of the problem, the accuracy represents the unique variable that needs to be optimized. It drives the electronics and is the primary purpose of the satellite for many mission phases. All the other variables and constraints are defined to better represent the system.

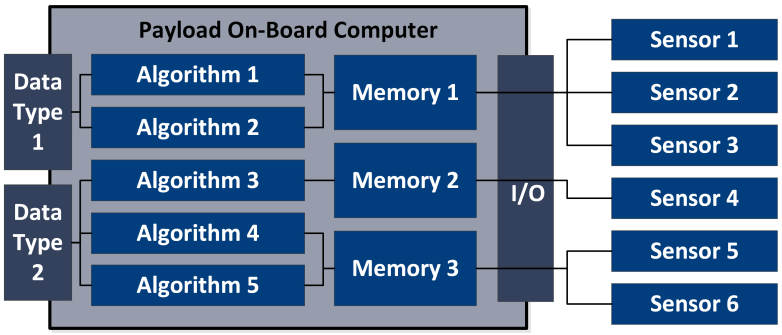


Figure 5.1 – **General model of avionic.**

2.1 Model Definition

As briefly explained above, the goal is to obtain the highest general accuracy on the target. To achieve this, the optimizer has the find the best mode sequence for all elements. The states represent the leverage of the optimizer to achieve the optimum solution. For this model, each element consists of binary variables assigned to a specific mode. By constraint, uniquely one variable can be set to 1 at the same time (Per timestep and per element.) A general model of the avionic is shown in Figure 5.1. It is simply a representation of the capability of the tool to use specific payload avionic architecture. For this chapter, a more simple one is used to better understand the behavior of the tool. It will be shown and described in the result section.

In this implementation, it is assumed that the architecture has a Payload On-Board Computer (PL OBC). This element consumes a certain amount of electrical energy and has some processing resources (MIPS) available for algorithms. As for all other components of the model, it contains various modes that each have defined parameters. For example, the Low Energy Consumption (LE) mode of the PL OBC consumes a reduced amount of electrical energy in comparison with the others, but merely a few processing resources are available.

In the PL OBC, there are also various algorithms with each a specific task. Each one contributes toward the determination of a type of data like the position of the target or the rotational rate. These algorithms are at a high level of abstraction and represent a potential implementation of an image processing or GNC process. It is assumed that they produce data for a type of data at a certain frequency. Nevertheless, multiple algorithms can contribute to a unique type of data with various levels. The algorithms' parameters are described later. To characterize them, the model defines a theoretical and abstract variable that stays between 0 and 1 and represents the accuracy of a data type. This abstract parameter allows the implementation of various algorithms without going into their specificity. Nevertheless, it adds a level of abstraction which can be problematic. This aspect is discussed later in the chapter. In addition, these algorithms contain multiple parameters that are mode-dependent. They consume processing resources from the PL OBC and data from the memories at a certain rate. They additionally have a specific frequency and various accuracy increments depending on their mode. As for the data type, the accuracy output of each algorithm is highly abstract. Finally, each memory

has one or multiple algorithms linked to them.

The memories remain reasonably classic elements. They receive data from various sensors and then deliver it to different algorithms. Their restrictions are in terms of storage capacity and data rate limit. Their purpose is primarily to do the interface between the sensors and algorithms and provide some additional constraints on the system. As for the previous chapter, the decision has been made to technically store all the information provided by the sensor. Even though it is probably not representative of a real mission, it allows testing the worst-case scenario in terms of data storage.

Next in the model, there are the sensors. Their goal is to provide information to the memories that can later be used by the algorithms. As for the PL OBC, they have an electrical energy consumption depending on their mode. These sensors also undergo a variation in data rates or production linked to their mode. The particularity is they include an "off" state where no information is produced and no energy consumed. As mentioned earlier, each sensor is linked to a unique memory, despite some receiving data from multiple components. As for the algorithm, the sensors' descriptions are highly generic and do not represent any specific components. Since the goal is to test the sensitivity of the tool, it was not required to implement precise components at this point.

A critical element is that each accuracy variable has an α parameter linked to it. It represents the loss of information over time. Indeed, when the system received an update from the algorithm the accuracy increases. However, the underlying trend of the variable is to decrease because of this parameter. The method is implemented to represent the effect of time in the system. The more time has passed since the update, the less confident the avionic is about the target. For example, the satellite can determine the exact relative position of the target at time t_0 and then use its navigation subsystem to propagate this information. Naturally, as the propagation is done on multiple occasions, the incertitude grows and the system is less confident about the result. The goal of the α parameters is to represent this behavior in a simple way.

For this model, it has been decided to implement the electrical energy generation of the satellite as a sinusoidal form with an offset. It can be perceived as if the satellite is including batteries that deliver a minimum amount of energy and then the solar panels provide the rest. In this case, the assumption implies the chaser is rotating around one of its axes and the electrical energy generated by the solar panels is fluctuating. It is a very generic way to represent the energy consumption of a satellite, nevertheless, it allows to apply tangible constraints to the system. The disadvantage of using electrical energy as a constraint is discussed later in the chapter.

Ultimately, the model also requires some initial conditions. The two used in this model are the initial accuracy on target and the memory usage. The second one is essentially barely used since it is more representative to put the actual data storage to 0 [Mb]. The time duration and the number of timesteps are other parameters of the simulation. It is significant to remember

that increasing the number of timesteps considerably raises the complexity of the model and thus the resolution time.

As a last notice, all the parameters are completely decoupled from the model itself in the simulation. It means it is possible to modify the payload avionic architecture implemented without touching the mathematical equations. To maintain more flexibility, the parameters are stored in Extensible Markup Language (XML) files while the model is written in Matlab. The advantage obtained is the simulation does not have to be modified to test new scenarios. The unique crucial point for the design is the definition of the parameters.

2.2 Variable Definition

All the variables below are defined for every iteration k of the simulation. They represent the elements the optimizer uses to determine the optimal solution. It is critical to remember uniquely the primary variables are truly controllable by the optimizer. The other variables are here to better represent the model and are highly constrained.

The first set of variables is here to describe the mode of the various elements in the payload avionic architecture. In this implementation, the algorithms, the sensors and the PL OBC need some control variable. For example, a sensor can be turned off or outputting a lot of data. As in the previous chapter, the usage of binary variables comes from the toolbox specificity.

Definition of the primary variable (Control variable):

- $u_{1,i,j}$: Binary variable for the mode j of the algorithm i.
- $u_{2,i,j}$: Binary variable for the mode j of the sensor i.
- $u_{3,j}$: Binary variable for the mode j of the PL OBC.

The variables shown below are here to describe the state or behavior of the system. Interestingly, some fundamental parameters such as the energy consumption are not represented here. These variables have the main purpose of storing information over time for the tool and not all physical parameters have to be stored.

A key part of this model is the accuracy of data type. As mentioned earlier, its goal is to represent the behavior of algorithms at a high level of abstraction. For this implementation, it is a way to estimate the efficiency of various algorithms used by an ADR mission. The algorithm can be image processing to find the target or even GNC process to compute the relative trajectory. Since a lot of variation exists between all the possibilities, the general and abstract accuracy variable is defined. It represents how much confident the system is about the target on specific aspects such as attitude or velocity. This data type accuracy is modified or updated by various algorithms. In this implementation, each algorithm makes a contribution to the variable at a certain frequency. This contribution has to be defined by

the user of the tool. A challenge is to match it to real value to have a representative mode. An image processing algorithm example could be to look at the precision of the process. This potential algorithm could deduce the position of the target with a 5% error. This number can then be transformed into an accuracy parameter for the algorithm model. The task to assess the level of accuracy required by the mission is given to the user of the tool. As for all these other parameters, the optimizer is uniquely providing their evolution over time with limitations due to the constraints and objectives. The designer has to implement the scenario and the actual constraint of the mission to obtain meaningful results. The accuracy aspect has some limitations that are presented in the challenge subsection.

Definition of the secondary variable (State variable):

- $x_{1,b}$: Continuous variable for the accuracy of the data type b.
- $x_{2,i}$: Continuous variable for accuracy increment of the algorithm i.
- $x_{3,i}$: Integer variable to store the last update time of the algorithm i.
- $x_{4,i}$: Binary variable to define if the algorithm i is updated.
- $x_{5,a}$: Continuous variable for the actual usage of memory a.

The last set of variables is more abstract. They have been defined to enable specific functionality in the tool.

Definition of the tertiary variable (Implementation variable):

- $y_{1,b}$: Binary variable to control that the accuracy is not smaller than 0 per data type b.
- $y_{2,i}$: Binary variable to control if the algorithm i has been updated (lower boundary).
- $y_{3,i}$: Binary variable to control if the algorithm i has been updated (upper boundary).

All the constants presented below are set at the beginning of the simulation. These are the parameters loaded from an external XML file. Most of these constants are defined for multiple elements of the system. Additional information is provided at the bottom of the list.

Definition of the constant:

- $U_{1,i}$: Algorithm i initial mode.
- $A_{i,j}$: Accuracy outputted by the algorithm i in mode j.
- $T_{1,i,j}$: Period of the algorithm i in mode j.
- F_i : Max period of algorithm i.
- $R_{1,i,j}$: Processing resources (or CPU load) used by the algorithm i in mode j.
- $D_{1,i,j}$: Data rate of the algorithm i in

mode j.

- $U_{2,i}$: Sensor i initial mode.
- $E_{2,i,j}$: Electrical energy consumed by the sensor i in mode j.
- $D_{2,i,j}$: Date rate of the sensor i in mode j.
- U_3 : PL OBC initial mode.
- $E_{3,j}$: Electrical energy consumed by the PL OBC in mode j.
- $R_{3,j}$: Processing resources (or CPU load) available in the PL OBC in mode j.
- α_b : Loss of information for data type b.
- $D_{4,a}$: Maximum data rate for the memory a.
- S_a : Maximum storage for the memory a.
- $I_{1,a}$: Initial storage used by the memory a.
- $M_{1,a}$: Algorithm ids linked to the memory a.
- $M_{2,a}$: Sensor ids linked to the memory a
- Y_b : Algorithm ids producing data type b.
- $I_{2,b}$: Initial accuracy per data type b.
- $E_{4,1}$: Electrical energy generated offset.
- $E_{4,2}$: Electrical energy generated amplitude.
- T_4 : Period of the electrical energy generated.
- N : Number of step in the simulation.
- dt : Timestep per iteration.
- m_1 : Number of algorithms.
- m_2 : Number of sensors.
- m_4 : Number of memories.
- m_5 : Number of data types.
- $l_{1,i}$: Number of modes for the algorithm i.
- $l_{2,i}$: Number of modes for the sensor i.
- l_3 : Number of modes for the PL OBC i.
- β_b : Weight for the objective function of accuracy data type b.

As general comments, the letter U is used for the initial mode of the elements. The letter R is for the processing resources or CPU load. The letter E is for the electrical energy aspect. The letter I is for the initial value of some parameters.

2.3 Quadratics Objectives

The objectives are in a quadratic form. k indicates the current step of the simulation.

The objective (Equation 5.1) implies that the accuracy on target $x_{1,b}$ should be maximal. To be noticed that each value is squared to depreciate the lower one. Moreover, the equation does not emphasize the accuracy value at any given time. The last point is the parameter β_b that defines the weight assigned to each type of data. It means the importance of each data type

respectively to the others. The equation considers the accuracy of each data type and for the whole scenario.

$$maximize \sum_{b=1}^{m_5} \beta_b * \sum_{k=1}^{N} \left[x_{1,b,k}\right]^2 \tag{5.1}$$

2.4 Constraints

In this part, the type of variable and the limit used in the model are defined. If the definition includes $\forall k$ it means for all iterations. (Exceptions are specified.)

The equation below can be divided into two categories of constraints. The first is the parameters representation or implementation constraints and the second one is the control constraints. The second group is required in order for the mathematical model to stay within the limit imposed by the system.

Equations 5.2, 5.3, and 5.4 define the limits of each variable. As shown in the equation 5.2, all the control variables are binary and they represent the current mode of each element of the system. Equations 5.5 prevent any elements to have more than one mode currently active at the same time and then equations 5.6 impose the initial state of the payload avionic.

Bound Constraints - The binary variable:

$$u_{1,i,j,k},\ u_{2,i,k},\ u_{3,j,k},\ x_{4,i,k},\ y_{1,b,k},\ y_{2,i,k},\ y_{3,i,k} \in \{0,1\} \quad \forall b,i,j,k \tag{5.2}$$

Bound Constraints - The integer variable:

$$x_{3,i,k} \in \mathbb{Z} \quad \forall i,k \tag{5.3a}$$

$$0 \leq x_{3,i,k} \leq N+1 \ \forall i,k \tag{5.3b}$$

Bound Constraints - The continuous variable:

$$x_{1,b,k},\ x_{2,i,k},\ x_{5,a,k} \in \mathbb{R} \quad \forall a,b,i,k \tag{5.4a}$$

$$0 \leq x_{1,b,k} \leq 1 \ \forall b,k \tag{5.4b}$$

$$0 \leq x_{2,i,k} \leq 1 \ \forall i,k \tag{5.4c}$$

$$0 \leq x_{5,a,k} \leq S_a \ \forall a,k \tag{5.4d}$$

Linear Constraints - The mode continuity constraint (only one mode can be active per item):

$$\sum_{j=1}^{l_{1,i}} u_{1,i,j,k} = 1 \ \ \forall i,k \tag{5.5a}$$

$$\sum_{j=1}^{l_{2,i}} u_{2,i,j,k} = 1 \ \ \forall i,k \tag{5.5b}$$

$$\sum_{j=1}^{l_3} u_{3,j,k} = 1 \ \ \forall k \tag{5.5c}$$

Linear Constraints - Initial mode constraint:

$$u_{1,i,j,1} = 1 \ \ , j = U_{1,i} \ \ \forall i \tag{5.6a}$$

$$u_{2,i,j,1} = 1 \ \ , j = U_{2,i} \ \ \forall i \tag{5.6b}$$

$$u_{3,j,1} = 1 \ \ \ , j = U_3 \tag{5.6c}$$

$$\tag{5.6d}$$

Equation 5.7 defines how the electrical energy in the system is produced and consumed. It is valid for every timestep of the simulation. The right side of the equation imposes the generation to be in sinusoidal form with a defined offset. It can be regarded as if the satellite is currently rotating and the solar panels are not oriented correctly toward the sun. The left part of the equation takes the energy consumption of all sensors and the PL OBC depending on their mode. Since uniquely one mode variable can be set to 1 per timestep, it means the sum operation retrieves the current consumption of all elements.

Linear Constraints - The electrical energy constraint:

$$\sum_{i=1}^{m_2} \sum_{j=1}^{l_{2,i}} u_{2,i,j,k} * E_{2,i,j} + \sum_{j=1}^{l_3} u_{3,j,k} * E_{3,j} \leq E_{4,1} + E_{4,2} * \sin(2\pi * \frac{k}{T_4}) \ \ \forall k \tag{5.7}$$

Equation 5.8 defines how the processing resources or CPU load in the system are handled and it is valid for every timestep of the simulation. This equation compares directly the processing resources available in the PL OBC with the ones needed for the algorithms. As previously, the resources are derived using the sum and the current mode of each element.

Linear Constraints - The processing resources constraint:

$$\sum_{j=1}^{l_3} u_{3,j,k} * R_{3,j} - \sum_{i=1}^{m_1} \sum_{j=1}^{l_{1,i}} u_{1,i,j,k} * R_{1,i,j} \geq 0 \ \ \forall k \tag{5.8}$$

Equation 5.9 defines the limitation of data rates for each memory of the model and is valid for

every timestep of the simulation. For every memory, the first sum is here to select all sensors linked to them. The second one is to extract the current data rate of the sensors depending on their modes. Ultimately, the sum of all the rates should be lower than an established fixed limit.

Linear Constraints - The data rate constraint:

$$\sum_{i \in M_{2,a}} \sum_{j=1}^{l_{2,i}} u_{2,i,j,k} * D_{2,i,j} \leq D_{4,a} \quad \forall a, k \tag{5.9}$$

Equations 5.10 define how the data storage is handled in the memories and it is valid for every timestep after the 1st iteration. At $k = 1$, the equation defines the current amount of data in the memory. For every other timestep, the previous usage is added to the current data rate of the sensors. (Multiplied by the time increment.) At that point, the usage of the algorithm is subtracted from the total. In this case, there is no inequation since it is a definition of the storage variable $x_{5,a}$.

Linear Constraints - The storage constraint:

$$x_{5,a,k} - x_{5,a,k-1} - dt * \sum_{i \in M_{2,a}} \sum_{j=1}^{l_{2,i}} u_{2,i,j,k} * D_{2,i,j} + dt * \sum_{i \in M_{1,a}} \sum_{j=1}^{l_{1,i}} u_{1,i,j,k} * D_{1,i,j} = 0 \tag{5.10a}$$

$$\forall a, \; k \in [2; N]$$

$$x_{5,a,1} = I_{1,a} \quad \forall a \tag{5.10b}$$

Equations 5.11, 5.12, 5.13 are defined in order for $x_{4,i}$ to be equal to 1 when the algorithm i is undergoing an update. It compares the actual period of the algorithm with the last time it was updated and the current timestep. The period is retrieved again by using the sum over the mode variable. This variable is then used by some other equations to update the states of the system.

Linear Constraints - Lower bound of the update variable constraint:

$$-x_{3,i,k-1} - \sum_{j=1}^{l_{1,i}} u_{1,i,j,k} * T_{1,i,j} + \delta * y_{3,i,k} \leq -k \tag{5.11}$$

$$\forall i, k \in [2; N] \; \& \; \delta \ll 1$$

Linear Constraints - Upper bound of the update variable constraint:

$$x_{3,i,k-1} + \sum_{j=1}^{l_{1,i}} u_{1,i,j,k} * T_{1,i,j-F_i * y_{3,i,k}} + \delta * y_{2,i,k} \leq k \tag{5.12}$$

$$\forall i, k \in [2; N] \; \& \; \delta \ll 1$$

Linear Constraints - Equality of the update variable constraint:

$$x_{4,i,k} + y_{2,i,k} + y_{3,i,k} = 1 \quad \forall i, k \in [2; N] \tag{5.13a}$$

$$x_{4,i,1} = 1 \quad \forall i \tag{5.13b}$$

Equations 5.14 serve as the definitions of each accuracy variable $x_{1,b}$ and are valid for every timestep after the first iteration. At $k = 1$, it is set to an initial value. The principle behind this equation is to limit the improvement of accuracy as it goes up. The equation can be rewritten as $x_{1,b,k} = x_{1,b,k-1} + \sum_{i \in Y_b} x_{2,i,k} * \left[1 - x_{1,b,k-1}\right] - \alpha_b * y_{1,b,k}$. By deduction, the actual accuracy variable is equal to the previous value in time with the addition of the current increment from the algorithm scale down. The idea behind this process is to limit the effect of updating an already high accuracy variable. In other terms, the gain of updating a data type with low accuracy is greater than with already high accuracy. The variable is also naturally decreased at every timestep by a factor alpha, but exclusively if $x_{1,b,k}$ stays above 0. Equations 5.15 & 5.16 are defined for this purpose exclusively. They ensure that the α decrease occurs exclusively when the accuracy variable is greater than 0.

Quadratic Constraints - Accuracy definition constraint:

$$x_{1,b,k-1} * \sum_{i \in Y_b} x_{2,i,k} + x_{1,b,k} - x_{1,b,k-1} - \sum_{i \in Y_b} x_{2,i,k} + \alpha_b * y_{1,b,k} = 0 \quad \forall b, \ k \in [2; N] \tag{5.14a}$$

$$x_{1,b,1} = I_{2,b} \quad \forall b \tag{5.14b}$$

Quadratic Constraints - Accuracy greater than 0 constraint 1:

$$-x_{1,b,k-1} * \sum_{i \in Y_b} x_{2,i,k} + x_{1,b,k-1} + \sum_{i \in Y_b} x_{2,i,k} - E * y_{1,b,k} \geq \alpha_b - E + \delta \quad \forall b, \ k \in [2; N] \ \& \ 1 \ll E \ \& \ \delta \ll 1 \tag{5.15}$$

Quadratic Constraints - Accuracy greater than 0 constraint 2:

$$-x_{1,b,k-1} * \sum_{i \in Y_b} x_{2,i,k} + x_{1,b,k-1} + \sum_{i \in Y_b} x_{2,i,k} - E * y_{1,b,k} \leq \alpha_b \quad \forall b, \ k \in [2; N] \ \& \ 1 \ll E \tag{5.16}$$

Equations 5.17 define when the last update happened for any algorithm of the system. The principle is to store the current iteration if an update happened, otherwise retain the previous value. It can be written as $x_{3,i,k} = k * x_{4,i,k} + \left[1 - x_{4,i,k}\right] * x_{3,i,k-1}$. It is also defined that an update happened at the first iteration $k = 1$.

Quadratic Constraints - Last update definition constraint:

$$x_{4,i,k} * x_{3,i,k-1} + x_{3,i,k} - x_{3,i,k-1} - k * x_{4,i,k} = 0 \quad \forall i, \quad k \in [2;N] \tag{5.17a}$$

$$x_{3,i,1} = 1 \quad \forall i \tag{5.17b}$$

Equation 5.18 sets the variable $x_{2,i}$ to the gain in accuracy of an algorithm if an update is happening, otherwise, the value is 0. This action is required to prevent the multiplication of three variables in the system.

Quadratic Constraints - Accuracy increment definition constraint:

$$-x_{4,i,k} * \sum_{j=1}^{l_{1,i}} u_{1,i,j,k} * A_{i,j} + x_{2,i,k} = 0 \quad \forall i,k \tag{5.18}$$

Equation 5.19 is here to prevent algorithms to change mode between updates. The modes would have to stay identical between the tool iterations if no update happened ($x_{k,i,k} = 0$). It can be written as $\left[\sum_{j=1}^{l_{1,i}} j * \left\{-u_{1,i,j,k-1} + u_{1,i,j,k}\right\}\right] * \left(1 - x_{4,i,k}\right) = 0$. If the update variable ($x_{4,i,k}$) is equal to 1, then this constraint does not exist ($0 = 0$).

Quadratic Constraints - Algorithm mode change constraint:

$$x_{4,i,k} * \left[\sum_{j=1}^{l_{1,i}} j * \left\{u_{1,i,j,k-1} - u_{1,i,j,k}\right\}\right] + \sum_{j=1}^{l_{1,i}} j * \left\{-u_{1,i,j,k-1} + u_{1,i,j,k}\right\} = 0 \quad \forall i, \quad k \in [2;N] \tag{5.19}$$

2.5 Challenges

The remaining issue with this recent model continues to be the high-level of abstraction. Validating or verifying such a model is relatively difficult and some effort will be necessary. The accuracy on target persists to represent the typical example that might need to be reworked. In the current implementation, it is purely a predetermined value produced by the algorithms depending on their mode. Ideally, these values would need to be extracted from actual hardware and software. Nevertheless, this abstract solution allows for an easier and faster implementation. In addition, it detaches the model from any specific algorithm implementation and permits testing of a broader range of elements without any tool modification.

As for the previous chapters, the memory's behavior can be discussed. At this point, all the information from the sensors is stored in the memories without any distinctions. Even though separation is done regarding the data type, the PL OBC stores everything in its memory. It means if no algorithm is running, then saturation can happen quickly. A straightforward solution could be to increase the size of the memory to prevent this. Nevertheless, a more realistic approach would be to store exclusively part of the sensor's information into these

memories. In addition, a behavior observes during the test phase is regarding the validity in terms of the data. In some simulations, the algorithms are using the information in the memory that is potentially outdated in a real flight scenario. It means the accuracy outputted by such algorithms should be impacted by the timestamp of the data produced.

A similar issue regarding this implementation is the electrical energy. In this model, it is assumed that the electrical energy aspect is uniquely optimized at the payload level. Nevertheless, some parts of the main satellite platform could potentially consume a magnitude more energy. It means constraining the payload avionic on the energy level during some phases could be inefficient.

The final issue exists in the quality of the produced solution. As it is explained in the following section, the result obtained does not seem to be optimum in all cases. Despite this model being more rigorous than in the previous version, some issues remain. Efforts have been invested in this model to prevent the optimization to be infeasible. Indeed, the design of the target accuracy prevents the value to be smaller than 0 and greater than 1 even if bounds constraints are removed. Other implementation tricks have been used to smooth the result. Nevertheless, it does not guarantee a more efficient solution at the end of the simulation. To have better outputs, it is vital to correctly define the parameters in the first place.

3 Optimizer Result

This section presents the results obtained thanks to the optimizer. The first part quickly described the architecture implemented. The second part looks at some of the classical results obtained. Ultimately, the last subsection is about parametric analyses of the architecture under various circumstances.

3.1 Mission Architecture

In this version of the optimizer, it has been decided to implement a reasonably classic architecture. The primary goal is to assess the capability of the tool and to analyze the effect of parametric change. The purpose was not to test a realistic payload avionic architecture. For the same reason, the architecture used almost matches the first one of the previous chapter. It has been slightly updated to match the capability of the tool.

Figure 5.2 shows the architecture used for this model. In this case, there are two single element chains in the design. It means the first sensor is connected to the first memory. At that point, this component is connected to the first algorithm. Ultimately, this element produces one type of data output. The second link is similar to the first one but with distinctive characteristics for both the sensors and the algorithm. The element that links the whole architecture together is clearly the energy needed to supply the PL OBC and the two sensors. It is in addition to the processing resources or CPU load provided by the PL OBC and needed by both algorithms.

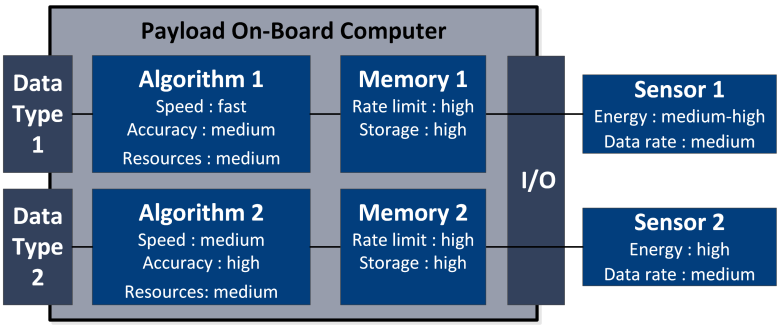


Figure 5.2 – **Payload avionic architecture implemented in the optimizer.**

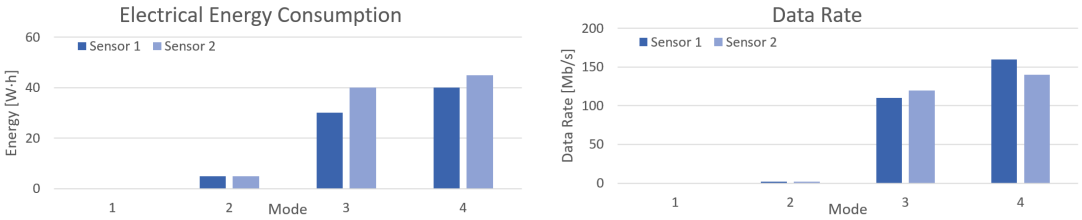


Figure 5.3 – **Sensors parameters.**

The idea behind this reasonably classic architecture is to decouple the maximum number of elements to better interpret their behavior. A simple avionic allows, in addition, to run faster optimization and therefore provides a broader range of results to analyze. This payload architecture allows creating a baseline regarding the sensitivity of the tool to some changes.

In the following paragraphs, a deeper explanation of the parameters for this specific architecture is provided. As a reminder, all the elements are kept to a high level of abstraction. The parameters shown are coming from a first-level of estimation and do not match any specific hardware or software components.

Figure 5.3 displays the two main parameters of both sensors. As it can be observed, each sensor contains four distinctive modes and the first one is considered as if the element is turned off. The second is frequently considered the LE mode since it uniquely consumes a little amount of energy and outputs almost no data. It can be envisioned as if a potential camera is taking pictures exclusively every few minutes. To conclude, there are the third and fourth modes which are relatively similar in both their electrical energy consumption and data rate. It can be observed that both sensors have also similar parameters in the order of magnitude. Regarding the energy consumption, the LE consumes about 5 $[W \cdot h]$ and the two advanced modes are around 35 $[W \cdot h]$. For the data rate, a few Mb/s are sent in LE. Nonetheless, it goes up to around 110 $[Mb/s]$ and 135 $[Mb/s]$ for the advanced mode.

Figure 5.4 shows all the parameters for both algorithms. Similar to the sensors, both algorithms contain four unique modes with the first one being the turn-off. Following a resembling trend, the LE is consuming a slight amount of processing resources and outputting almost

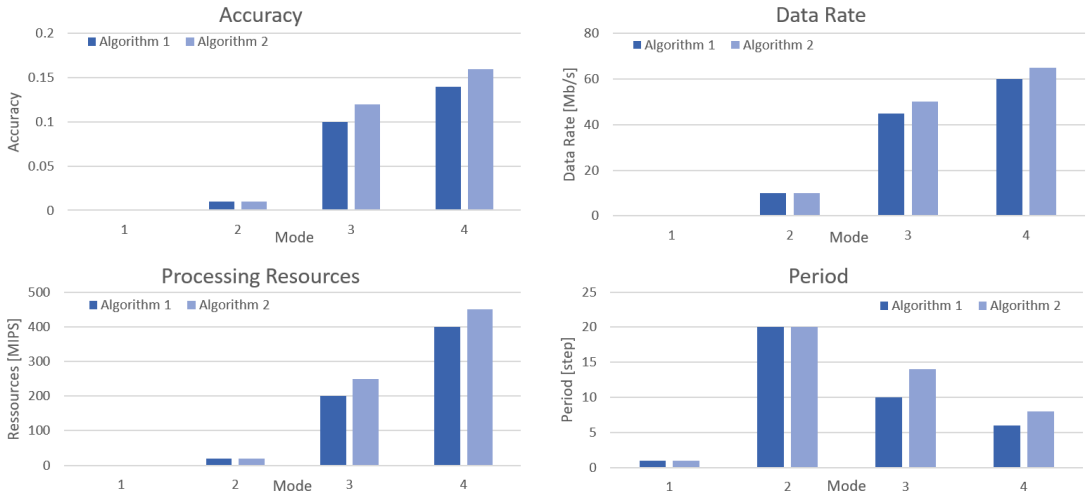


Figure 5.4 – **Algorithms parameters.**

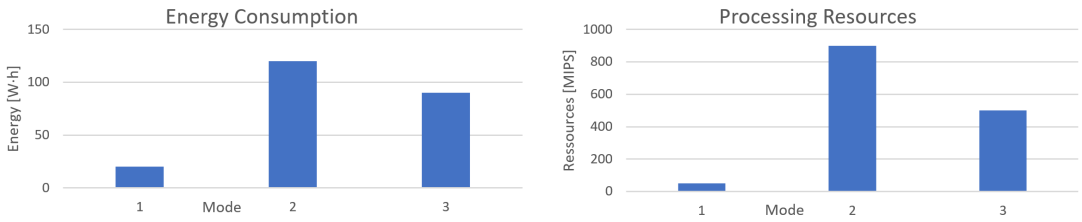


Figure 5.5 – **PL OBC parameters.**

no accuracy increment. In closing, there are the third and fourth modes which are again similar for most parameters. Regarding the accuracy, the fourth maintains better efficiency than the third one with respectively around 0.15 and 0.11. The LE falls to less than a tenth of these values with 0.01. For the data rate, the fourth mode is slightly more data-consuming relatively the third one with 60 $[Mb/s]$ versus 45 $[Mb/s]$. The LE is directly set at 10 $[Mb/s]$. The following parameter represents the processing resources consumed by the algorithms. In this case, the fourth mode required practically twice the third one with around 400 $[MIPS]$ against 200 $[MIPS]$. In comparison, the LE needs 20 $[MIPS]$. The last algorithm parameter is the period at which they are running. It can be observed that the power-off mode undergoes the lowest period, however, it is unimportant since no output is produced. The fundamental point is to allow the optimizer to turn on the algorithm quickly if needed. Comparatively, the LE has a period of 20 $steps$ which means the system can be stuck for a relatively extended time in this mode. Ultimately, the third is in the range of 14 $steps$ and the fourth is around 7 $steps$. There is a trade-off between modes three and four. The latter has better accuracy and faster computational time however it consumes more processing resources and requires additional data. It is assumed to be the high-demanding mode with a complex implementation relatively to a more ordinary and classic version with the third.

Figure 5.5 shows the two parameters of the PL OBC. The primary difference with the previous elements of the architecture is the PL OBC has no power-off mode meaning its first one is the LE. Indeed, this function was removed since it was not required in the optimization. The first mode utilizes a minimal amount of electrical energy and few processing resources are available. It is a typical case of sleeping mode. For this element, the second supports the high-efficiency task and the third one represents the ordinary case. Regarding the energy, the LE consumes around 20 $[W \cdot h]$. In comparison, the second goes up to 120 $[W \cdot h]$, and the third is at 90 $[W \cdot h]$. For the processing resources, there are exclusively 50 $[MIPS]$ available in the sleeping mode, and the high efficiency reaches 900 $[MIPS]$. In comparison, the standard mode is in-between with 500 $[MIPS]$ available.

Another element of the architecture is the memory. To stay with a relatively straightforward design, the memories do not include any mode and have a maximum storage and data rate. For both of them shown in Figure 5.2, they have 200 $[Mb/s]$ bandwidth limit and a capacity of 1000 $[Mb]$ each. It is assumed that their energy consumption is included in the PL OBC. At the beginning of the simulation, it is established that the two memories are empty. In addition, both types of output data have an alpha parameter of 0.002. It means the accuracy on the target decrease by this factor at every timestep.

Regarding the avionic architecture, the essential point to mention is the electrical energy generation. As explained in the mathematical model section, the energy is defined by a minimum offset with a sinusoidal variation. In the ordinary case, the offset is set to 180 $[W \cdot h]$ with an amplitude on the sinus of 80 $[W \cdot h]$. The period is fixed to 40 $steps$.

Finally, the number of steps is set to 80 for the simulation to obtain rapid results and be able to analyze the effect of various parameters. It is a trade-off between the precision of the tool and the length of the optimization run. The step duration is technically not imposed by the implementation. For this chapter and the following, it is assumed that one step is equivalent to 2 $[s]$. It means all the scenarios represent a phase of 160 $[s]$ of avionic payload operation. It is obviously a short duration, but it allows fast iteration of the design. As a side note, the weight assigned to both accuracy on target parameters in the objectives function is simply 1.

3.2 Reference Scenario

The goal of this section is to analyze in detail the result of the optimizer for one specific scenario. It is a crucial step before comparing the effect of parameter changes in the following subsection. In this case, the parameters presented above are used without any modification. It produces an architecture with two chains including each: one sensor, one memory, and one algorithm. The optimizer is run over 80 steps and needs to seek the best way to operate the mode of the various elements. The analyses of a specific scenario are ordinarily performed with the help of multiple plots. For this application, it has been decided to look at the evolution of the mode changes, accuracy, memory usage, power, processing resources, and data rate at the memory. This scenario focuses on a variation of electrical energy available in the system.

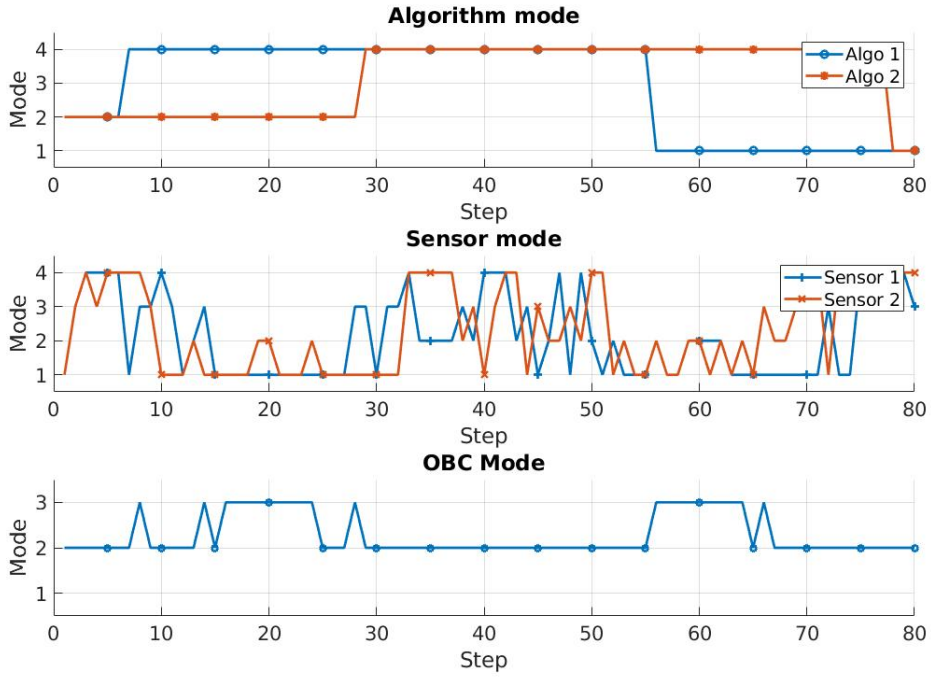


Figure 5.6 – **Reference scenario 1 - modes.**

This variation can be linked to a potential rotation of the spacecraft and thus variation in the solar panels' illumination. It is, nonetheless, a simple scenario that lacks the link to a real flight phase of an ADR mission. The goal of staying at this level of abstraction is again to test the capability of the tool with exclusively moving constraints on the payload avionic architecture.

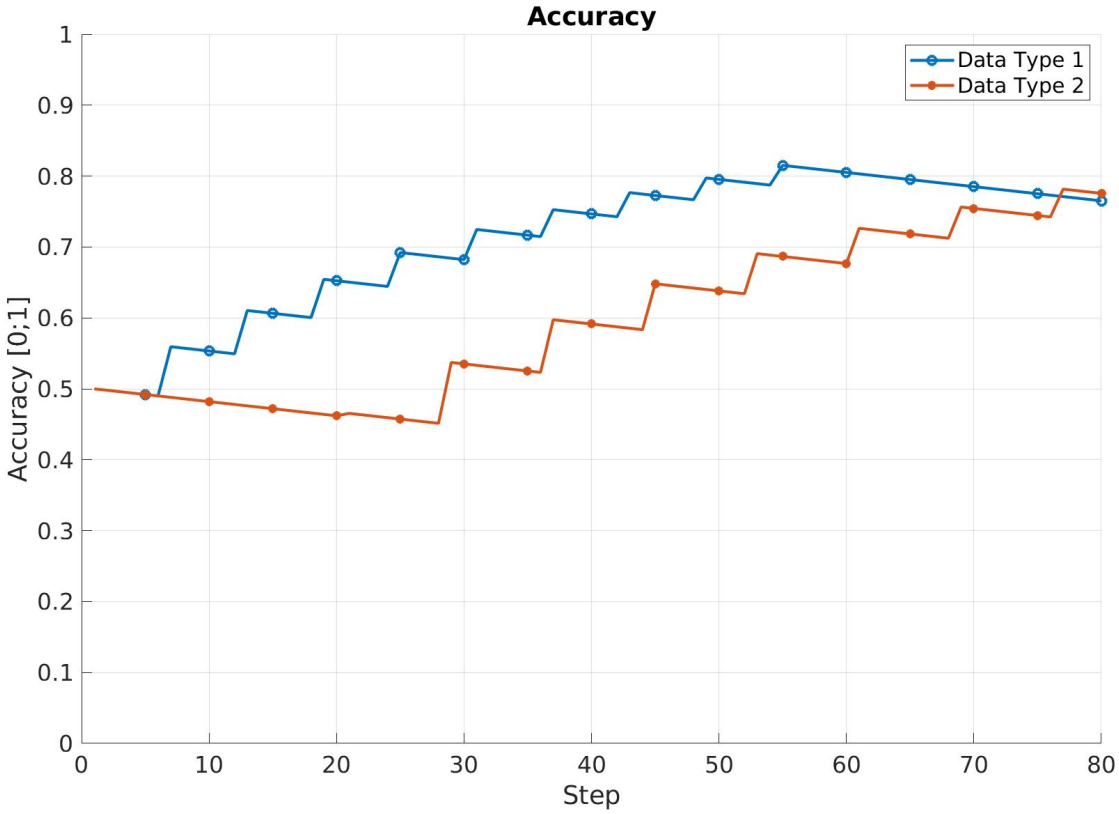
By looking at Figure 5.6, the variation in modes is an excellent starting point for the analyses. It can be noted that the PL OBC alternates between modes three and two. As a reminder, the second one remains the most prevailing one. It will be correlated later in this section, but the second mode is happening when the system is not under tight energy constraints. Regarding the algorithms, the first one is promptly set to mode 4 (high-efficiency) until two-thirds of the simulation when it moves to power off. Distinct behavior can be observed for the second algorithm. To begin with, it stays in the LE mode for one-third of the simulation and then is set to high-efficiency (mode 4). It is indicated again later, but part of the behavior is directly linked to the processing resources available and thus the energy the PL OBC can use. Furthermore, there is an influence on how the objective function is modeled, but this phenomenon is explained later. The last part is about the two sensors. Their operations vary a lot overall but still follow the trend of the energy. Their quick variation can be explained by the lack of constraints regarding their mode's transition. This behavior is not wanted since it is not a realistic case. A straightforward solution would be to constrain their mode change to a minimum duration, however, it is not the case in this version of the optimizer. Nonetheless, it is decisive for the system to maintain them working, so the data required by the algorithms

are produced.

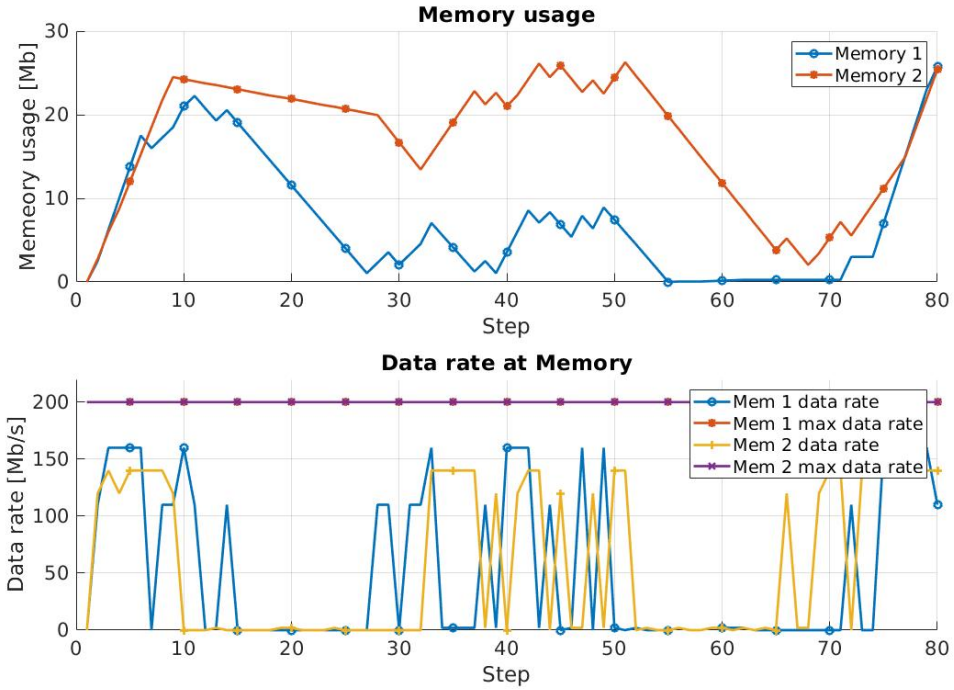
Figure 5.7a shows the evolution of the accuracy over time. The output one has first an increase and then stops being refreshed. In opposition, the second output is first left without an update before receiving an increase until the end of the simulation. The first thing to observe is that the no update part for the first and second data types happens uniquely when the energy available is tight (Figure 5.8a). As mentioned in the past, it is linked to the PL OBC changing to a less powerful mode and therefore having fewer processing resources available for the algorithms. Regarding the switch between data types one and two, it originates from the objective function. Since the weight assigned to both outputs is the same, the objective could have picked exclusively one of them and discarded the other one. Nevertheless, it is significant to remember that the function is quadratic meaning that both outputs are squared at every iteration. Considering the objective function wants to achieve the highest value, the update of lower accuracy is favored since it provides a greater gain. For this reason, the second output is updated at the end of the simulation and not the first one. Regarding the general shape of the accuracy parameter, it is important to understand the meaning of the steps. This effect happens every time an algorithm produces an output that is linked to its internal frequency. This update is applied to the data type variable and thus the steep increase. The rest of the time the variable is simply decreasing due to the natural loss of accuracy over time. This effect is the direct consequence of the α parameter.

Figure 5.7b shows the memory usage and the data rate at memory over time. It can be noted that both elements have different usage. The second one is overall holding more data than the first one. It is due to the absence of a second algorithm update in the first phase of the simulation. It means the data are gradually accumulated. Nevertheless, a similar trend can be observed for both memories regarding the increase and decrease of data stored. For both, the increase is directly linked to the mode of the sensor and thus the energy available. Nevertheless, it is critical for them to provide enough data, so the algorithms are not restrained in their task. Regarding the data rate, it shows the link between the memories and the sensors. In this case, the correlation with the electrical energy is obvious by looking at the period when nothing is happening. In this period, almost no energy is available for the sensor and thus almost no data is produced. In this design, it has been decided to not limit the bandwidth of the sensor for simplicity. The maximum data rate for both memories is at the same limit of 200 $[Mb/s]$.

Figure 5.8a shows the evolution of electrical energy consumption and production over the full simulation. The most distinctive part is the generation with a clean sinus waveform. In this case, the production starts with the peak before decreasing. The other thoroughly interesting trend is the one from the PL OBC. It can be identified that it reaches several times a point when the total available energy is insufficient to maintain the PL OBC in its high-efficiency mode. This limitation is not ideal and its implication is discussed in the latter part of the thesis. Regarding the mode change, it causes a direct effect on the processing resources available and thus on which algorithm can be run. The latter valuable point is, of course, the variation



(a) Accuracy of output data.



(b) Memory usage and data rate at memory.

Figure 5.7 – **Reference scenario 1 - Part 1.**

happening with the two sensors. In their case, they follow more or less the data demand at the memory. The decisive part is the PL OBC seems to be generally prioritized over the sensors. Indeed, the data generated by the sensors are accumulated for later use, but the PL OBC possesses instantaneous processing resources. In addition, the objective function is based directly on the output of the algorithm which depends on the processing resources. The algorithms using old information is again an issue.

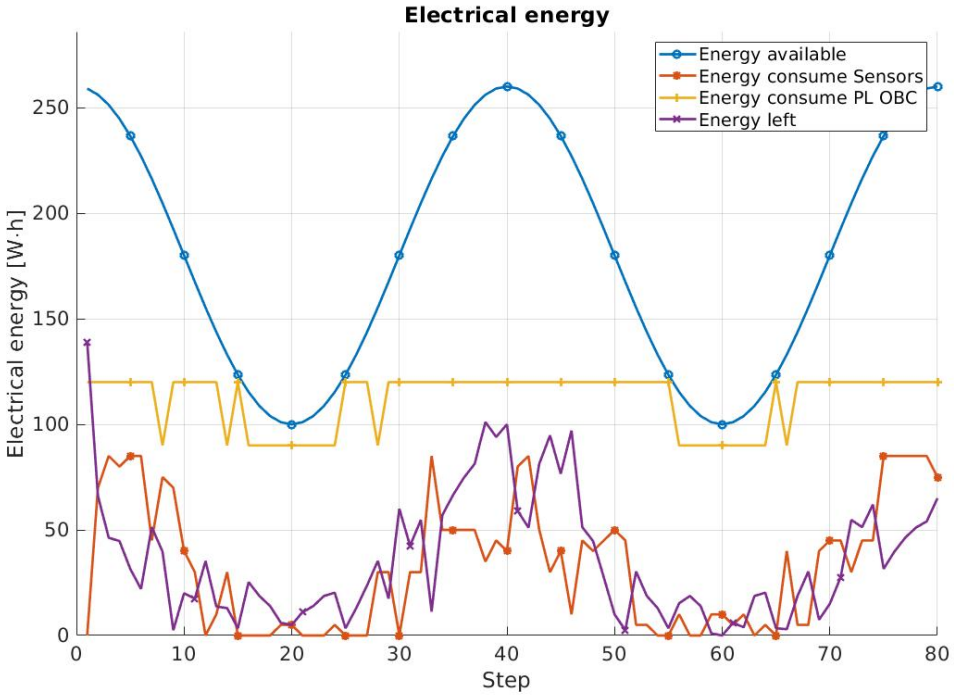
Figure 5.8b shows the processing resources or CPU load in the payload avionic. The comparison is performed between what is available in the PL OBC and what is consumed by the algorithm. Key points to recall are algorithms utilize processing resources that differ a lot and they are constrained by their period for mode switching. It means gaps linked to these constraints can exist in the plot. It can be noted that the processing resources consumed create a staircase shape directly related to the high-efficiency mode of both algorithms. As explained before, there are one then two, and one again algorithms in full mode. The two of them using almost the whole processing resources can merely happen when the electrical energy is at its maximum. Some tough needs to be done to comprehend why in the terminal part of the simulation exclusively one algorithm is running. The reason might be the modest amount of data present since the sensors are practically turned off during the low-energy phase. Moreover, the accuracy is already high for the first type of data, thus the optimizer might have chosen not to employ the algorithm.

3.3 Parametric Analysis

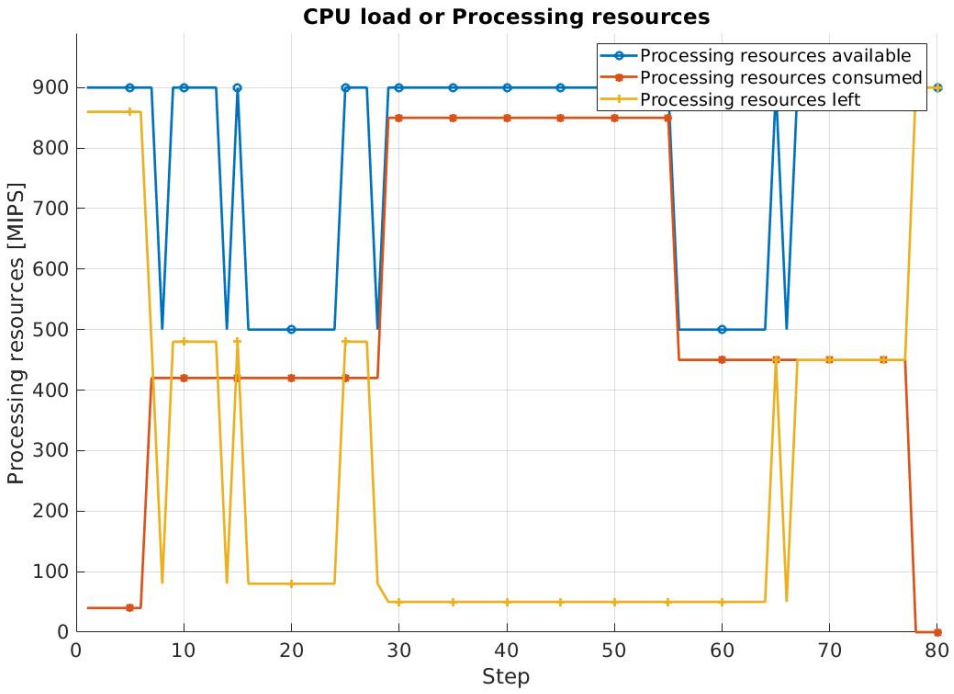
In this subsection, the goal is to assess the effect on various parameters with parametric analyses. The main approach is to look at the effect of a parameter change in the accuracy over time. It allows the evaluation of the accuracy variable's sensitivity in the tool to some changes.

The basic setup of the optimization stays the same as presented in the previous subsection. It means all the parameters are fixed to the default one except for one varied with a predetermined interval. For these analyses, it has been decided to select various parameters like the amplitude, offset, and frequency of the electrical energy generation. In addition, inspection is conducted on the alpha factor, the processing resources of the PL OBC, and the weights of the objective function. All these comparisons are ordinarily performed with seven different scenarios where the number four has habitually the default parameters. Every time, the accuracy parameter and another one are shown to evaluate the effect of the changes. In all the plots presented below, the markers on each curve possess no specific meaning. It has been placed to enhance the readability. An important to remember is that these scenarios are not linked to any specific flight phase. Their designs have been done specifically to test the sensitivity of the tool. With these analyses, it is then possible to model realistic elements with emphasis on the sensitive ones.

To start, an energy amplitude comparison is done with exclusively two variations. The goal is to explain in greater detail the implication of a specific change in the constraint. Figure 5.9 shows



(a) Electrical energy.



(b) Processing resources.

Figure 5.8 – **Reference scenario 1 - Part 2.**

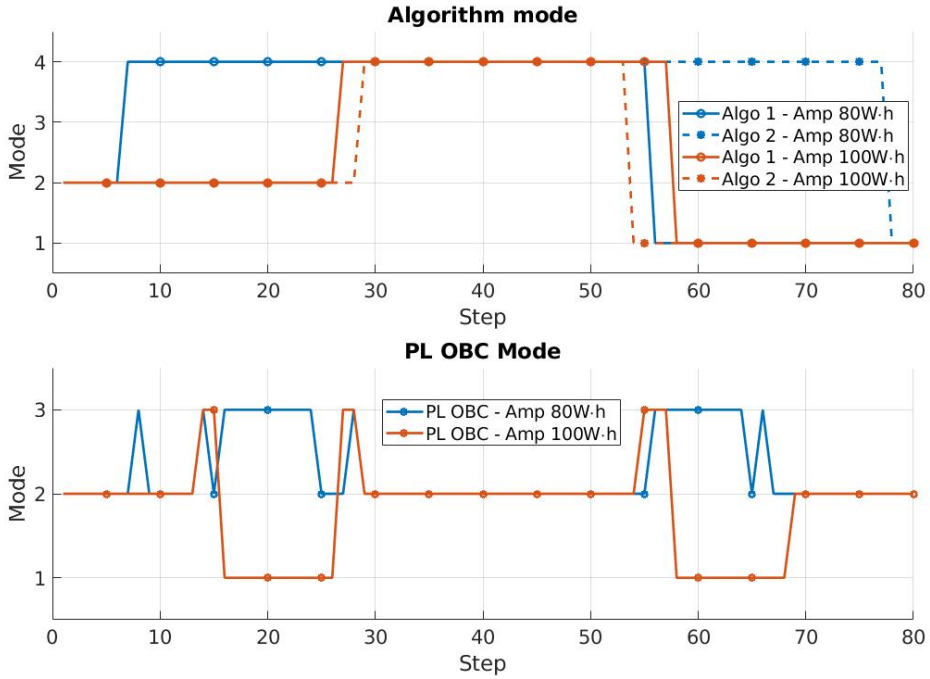
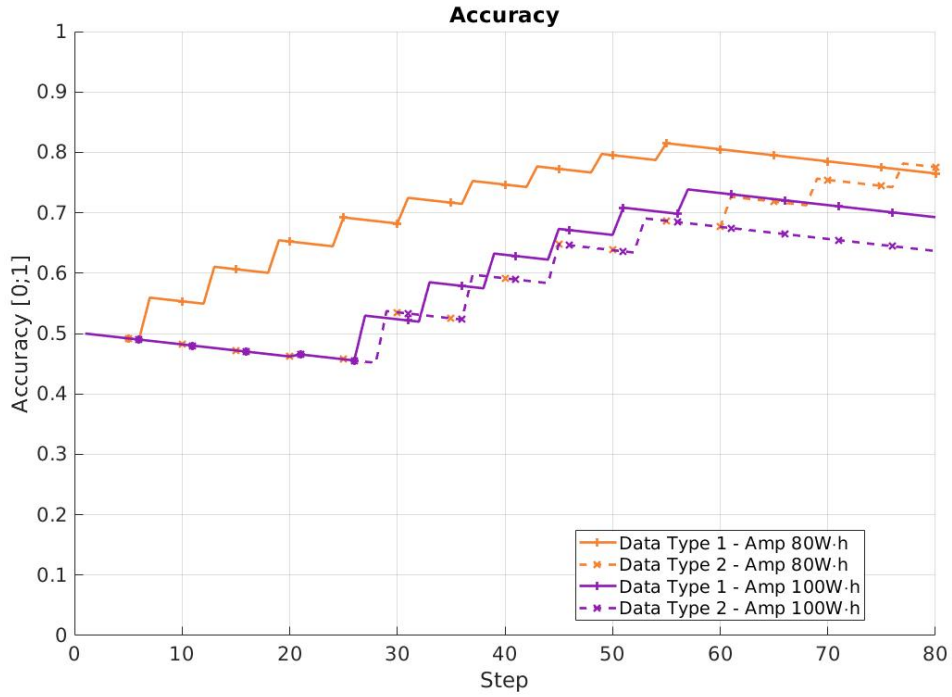


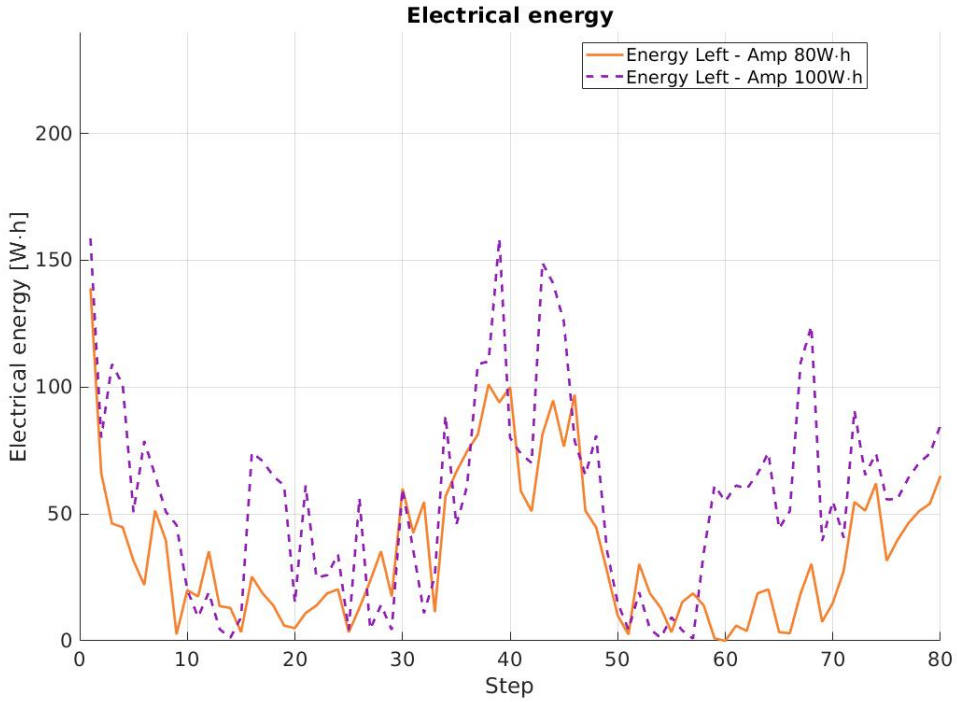
Figure 5.9 – **Electrical energy amplitude comparison (simple) - modes.**

the mode of the same architecture with two constraints on the amplitude of the electrical energy. The amplitude is set to 80 $[W \cdot h]$ and 100 $[W \cdot h]$ respectively. An important point is that the two curves uniquely show the electrical energy left in the system. As mentioned before, the energy supply is the same as in the reference scenario but with a different amplitude. It can be observed that the mode variation differs slightly between the two cases. In the second case, the second low energy phase implies that the PL OBC has to be put in LE mode. By consequence, both algorithms can not be used and are turned off. The electrical energy consumption can be seen in Figure 5.10b. Even though the high phase has more electrical energy, the low phase is longer. It is for this reason that the PL OBC is set to LE. Figure 5.10a display the differences in accuracy for both cases. The result is pretty obvious when linked to the mode switching. Since the algorithm can not be used as often in the second case, the overall accuracy is lower.

For the full energy amplitude analyses, the range is to move from 0 $[W \cdot h]$ to 120 $[W \cdot h]$ with a 20 $[W \cdot h]$ step. It signifies the first scenario has almost no fluctuation and the last is subject to drastic changes. Linking this to a real flight scenario, it could be related to the efficiency of the solar panels. It means this analysis could potentially be looking at trade-offs of solar cell type and see its implication on the overall accuracy parameter. In other terms, does the electrical energy production affect directly the efficiency of the rendezvous algorithms? For this analysis, the behavior of the accuracy and the electrical energy is shown. It was decided not to display the other variable for clarity purposes. Looking at Figure 5.11a, it can be observed that the influence of the fluctuation starts around scenario 5 which means an amplitude of 80 $[W \cdot h]$.



(a) Accuracy.



(b) Electrical energy left.

Figure 5.10 – **Electrical energy amplitude comparison (simple).**

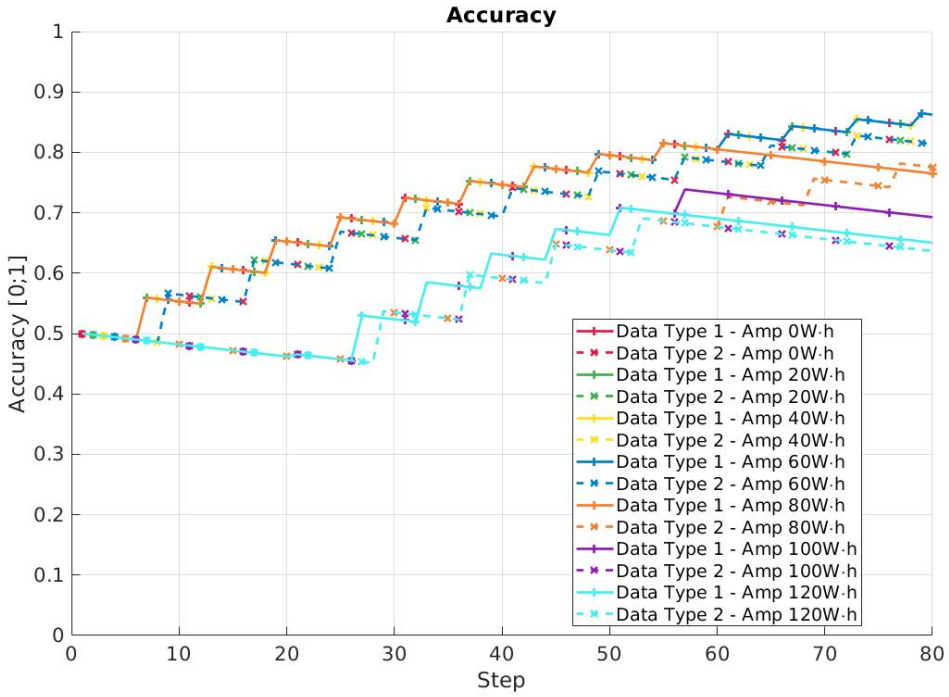
For all of them with a lower amplitude, the accuracy result is identical.

By observing Figure 5.11b, the trend of energy fluctuates a lot. In this case, uniquely the electrical energy left in the system is shown and thus the perfect sinus generation can not be seen. For the scenarios with large energy amplitude, the avionic has a substantial amount of electrical energy since the whole system is already at maximum efficiency. Nevertheless, the avionic is struggling during the low phase with close to 0 $[W \cdot h]$ left for the last scenarios. This shortage effect is observed in the accuracy drop since no algorithm can be run.

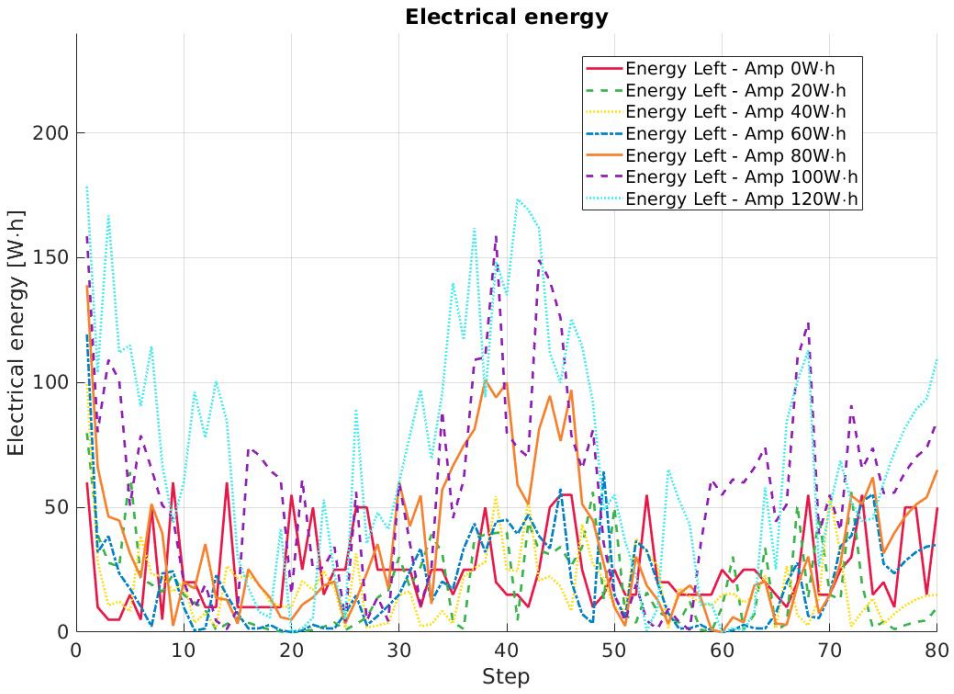
Regarding the energy offset, the variation is from 100 $[W \cdot h]$ to 300 $[W \cdot h]$ with a step of 40 $[W \cdot h]$. This analysis could come from a trade-off between multiple batteries' sizes. The goal is to look at the effect of this variation on the accuracy variable. In this case, exclusively six scenarios are generated. In reverse from the previous parametric analyses, the first up to the fourth one are affected by the offset modification. In Figure 5.12a, scenarios 4 to 6 have the exact same output. It means an energy offset equal to or above 220 $[W \cdot h]$ is enough for the avionic to run at maximum efficiency. This value is directly linked to the consumption of the various elements in the payload avionic. Nevertheless, it indicates that at a certain level, increasing the battery size (or minimal offset) does not affect the performance of the design.

Nonetheless, Figure 5.12b shows that scenario 4 is still reaching 0 $[W \cdot h]$ left in few occasions. By altering the sensors mode, the avionic can adapt and preserve the algorithms. It is no longer the case for below 220 $[W \cdot h]$ as the accuracy stops being updated at some point of the low energy phase. This expected behavior happens since a low energy supply implies the PL OBC can not allow sufficient processing resources for the algorithms. It is noted that the general result is relatively similar to the previous parametric analyses. From a designer's point of view, a 0 $[W \cdot h]$ is not acceptable. Some margin would need to be taken into consideration to obtain a proposer design. Nevertheless, the tool provides the needed information to tackle this issue.

The next parametric analysis is the effect of the electrical energy period. As a comparison, this variation can be linked to the rotational rate of the satellite around one of its axis. For example, the chaser needs to synchronize its rotational rate with the target it is trying to catch. It would mean the solar panels receive some solar energy at various rates assuming they have a fixed orientation. The goal of this analysis is to look at the consequence of this behavior on the accuracy parameter. For this set of simulations, the period of the energy fluctuation is varied between 10 $steps$ and 70 $steps$ with an increment of 10 $steps$. In this case, the result is substantially harder to read since the peak starts consistently at step 1 of the simulation. It means high period scenarios spend more time in the low energy phase. In Figure 5.13a, multiple different behaviors can be observed. Scenario 6 and 7 both have data type 1 which is the highest but the number 2 is the lowest. On the opposite, scenarios 1 and 2 are more in the middle range with still some lack of update for one of their algorithm. Overall, the fourth and fifth seem to remain the ones that can achieve relatively high accuracy for both their output. Interestingly scenario 4 produces both curves close to each other while they are more spread in the fifth.

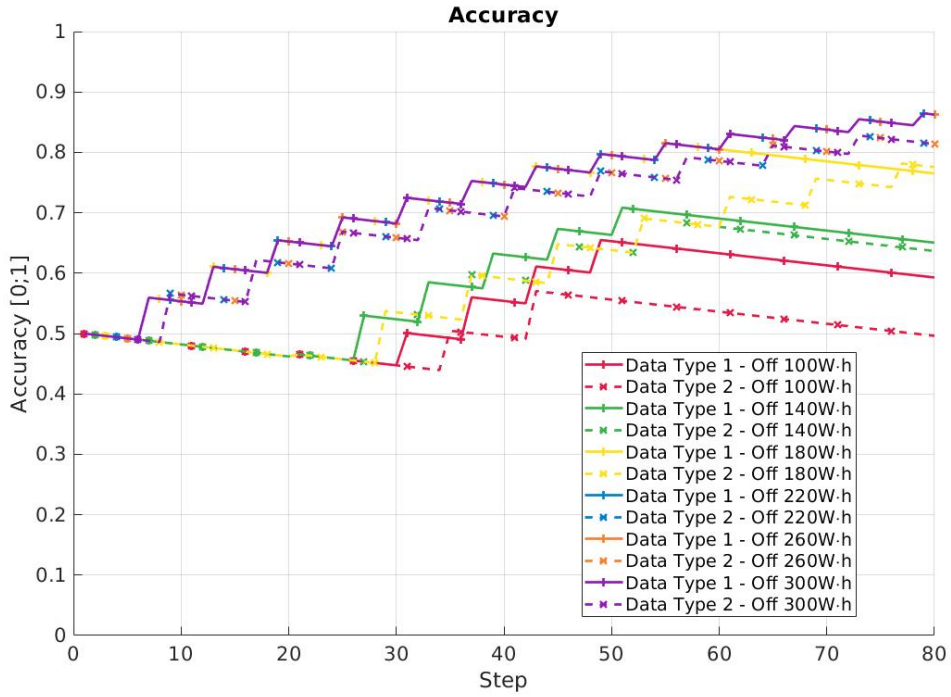


(a) Accuracy.

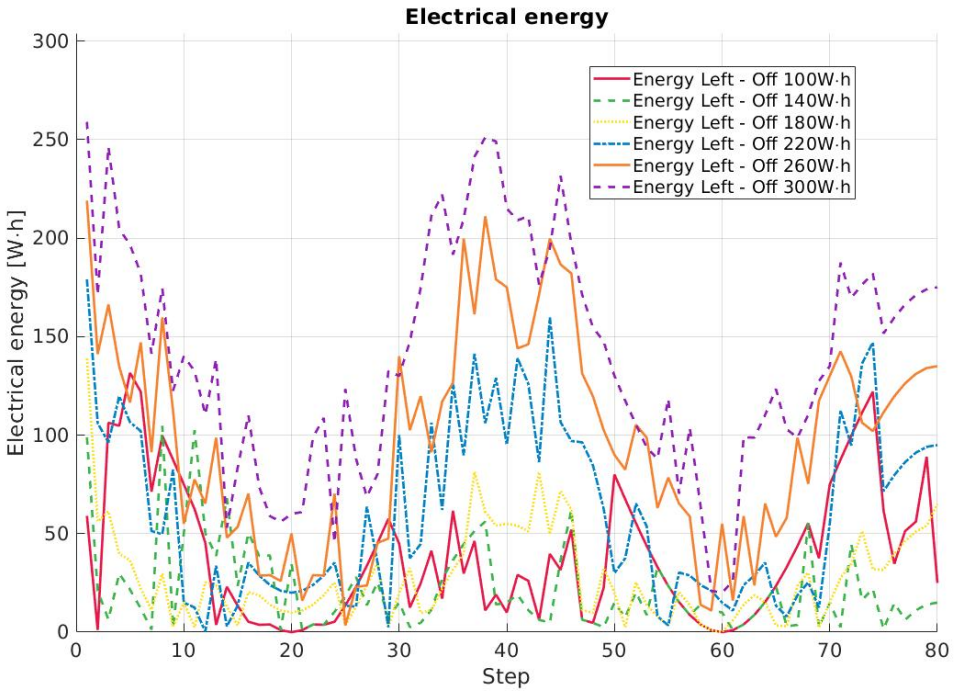


(b) Electrical energy left.

Figure 5.11 – **Electrical energy amplitude comparison.**



(a) Accuracy.



(b) Electrical energy left.

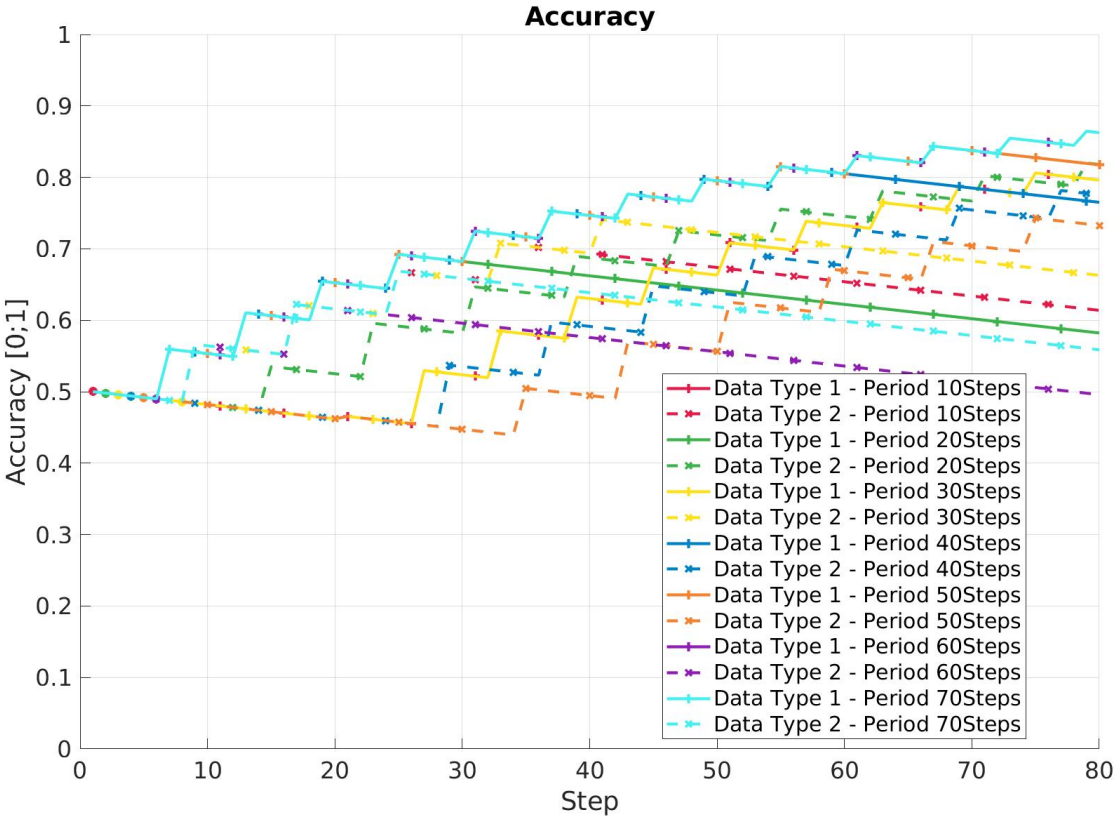
Figure 5.12 – **Electrical energy offset comparison.**

Figure 5.13b shows the difference between scenario 1 which contains multiple short peaks of power and the seventh which includes a large one around the end of the simulation. (For clarity purposes, uniquely scenarios 1,4, and 7 are shown with the electrical energy left in the payload avionic.) It is crucial to remember that algorithms experience periods that have an equivalent length as the energy in the first scenario. It means they can repeatedly not be run at maximum efficiency since the PL OBC is not staying in the same mode during the period. For these reasons, undergoing quick variation in the electrical energy has a negative effect on the output. Relatively, the large oscillations are also penalized because of the long low energy phase. It implies the algorithm can not be run, and the loss of information causes a vaster effect. The compromise seems to be found in the mid-range of the scenario. Nevertheless, increasing the size of the battery (as seen before) and thus the minimal energy supply would solve most of these issues.

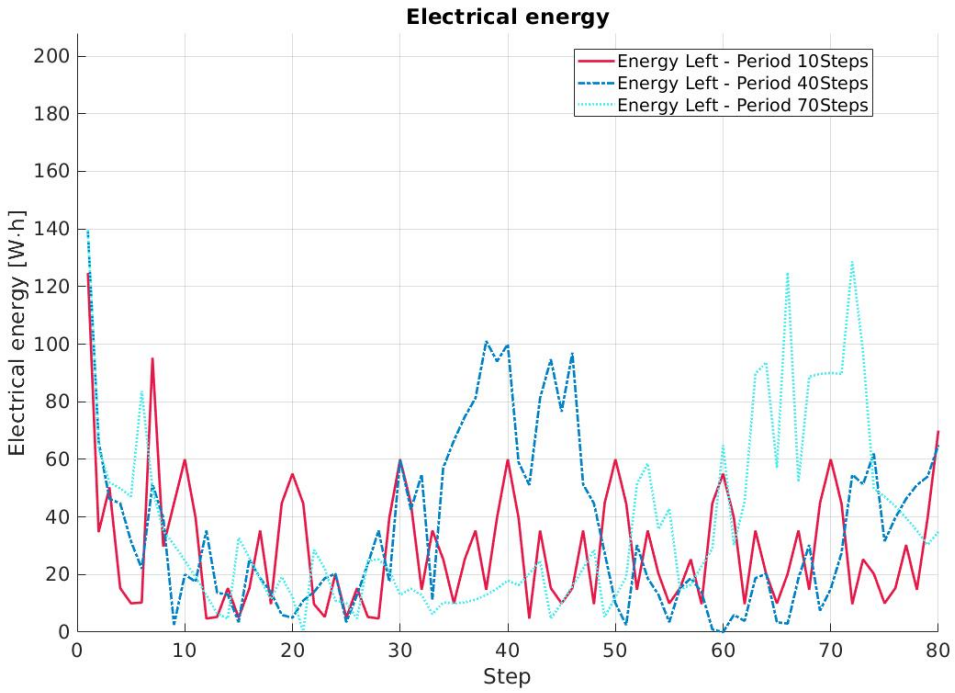
For the next parametric analysis, the electrical energy constraint is left and moved to another type of parameter. The first analysis is about the effect of the alpha parameter on the optimizer. As a reminder, this parameter affects how much information is lost over time if there are no algorithm updates. In other terms, is it the decreasing slope of the accuracy parameter? For this case, the variation goes from $1 * 10^{-3}$ to $4 * 10^{-3}$ with a step of $5 * 10^{-4}$. As mentioned earlier the regular case is at $2 * 10^{-3}$ and a higher alpha parameter means a greater loss of information over time. This analysis is useful to determine the importance of this parameter on the overall optimization. Since it is highly abstract, sensitivity is essential to understand its implication. Figure 5.14a shows the expected result with the lower alpha parameters scenario growing faster and staying higher than the other. It is intriguing to notice that this change is not affecting the simulator regarding which algorithm to run and when. On average, the trend is comparable to the one explained in the previous subsection.

Figure 5.14b confirms the assumption made by showing little variations in the energy left in the avionic. These slight variations can be explained by the convergence of the algorithm toward an optimal solution and the lack of constraint on the sensors' usage. Predominantly, it can be deduced that the alpha parameters exclusively produce an impact on the data type output. No effect is observed on the general behavior of the optimizer. It is promising since the parameter is difficult to match to a concrete measurement. It means an error in this design has little impact on the final result.

This parametric analysis is slightly independent of the other one. In this case, it has been decided to vary the processing resources available at PL OBC for its second and third modes. This is a useful analysis if a trade-off regarding the PL OBC capability is required. The task is to understand the implication of the variation on the overall accuracy parameter. To better interpret the effect, a percentage of the ordinary case is handled with a starting point at 40% until 160% with an increment of 20%. Figure 5.15a shows once again the evolution of the accuracy over time. The first thing that can be noticed is that the result for scenarios 6 and 7 are identical. It is similar for the fourth and fifth. These four scenarios seem to have approximately the same overall output. A decisive point to recall is the PL OBC is still subject to the regular

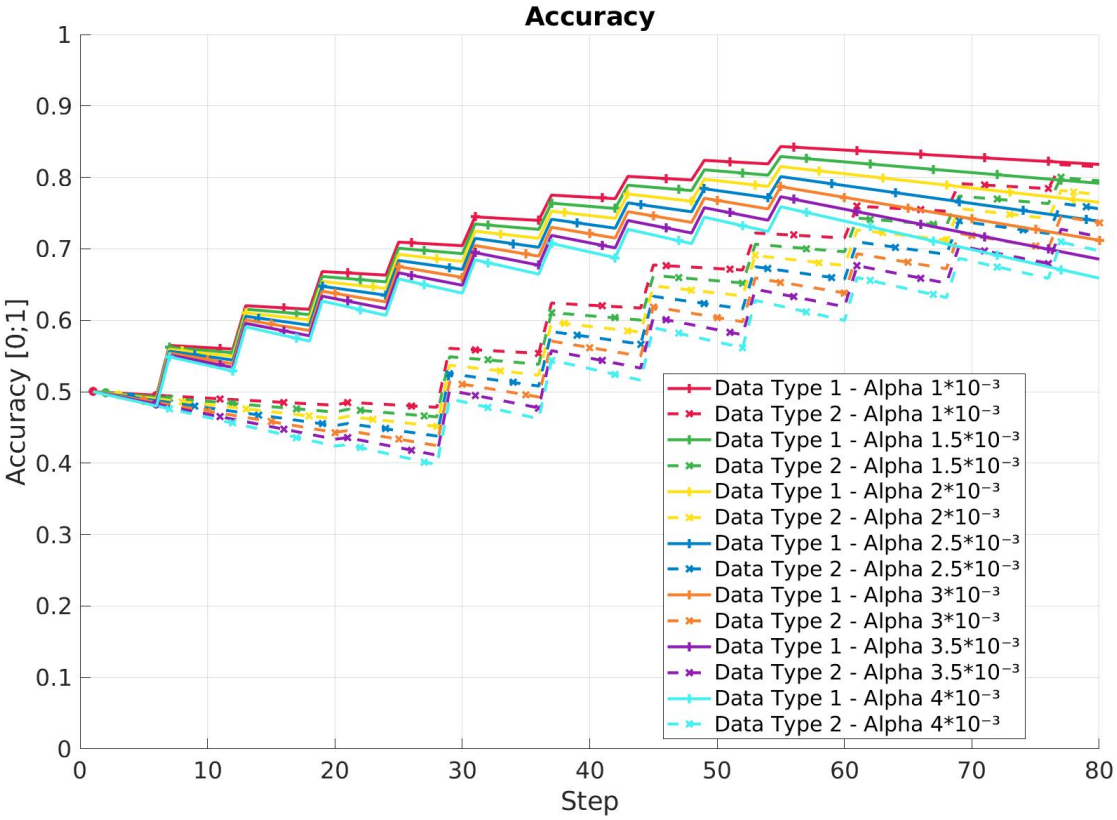


(a) Accuracy.

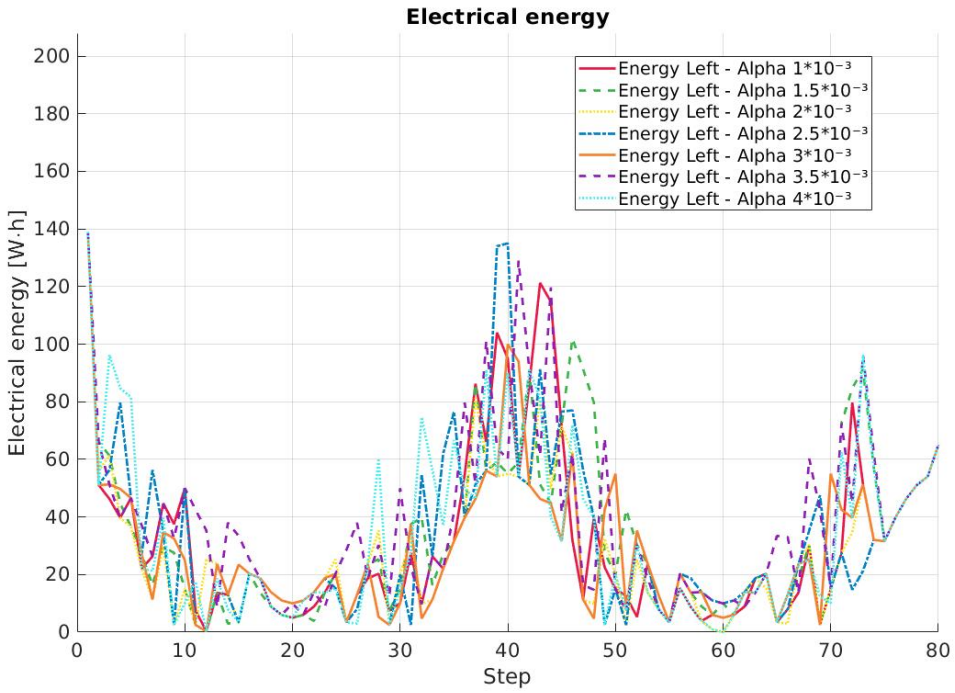


(b) Electrical energy left.

Figure 5.13 – **Electrical energy period comparison.**



(a) Accuracy.



(b) Electrical energy left.

Figure 5.14 – **Alpha comparison.**

energy constraint. As consequence, it is impossible to stay in the high-efficiency mode during the whole simulation. Nevertheless, increasing the capability without modifying the electrical energy usage means more flexibility in the algorithm usage.

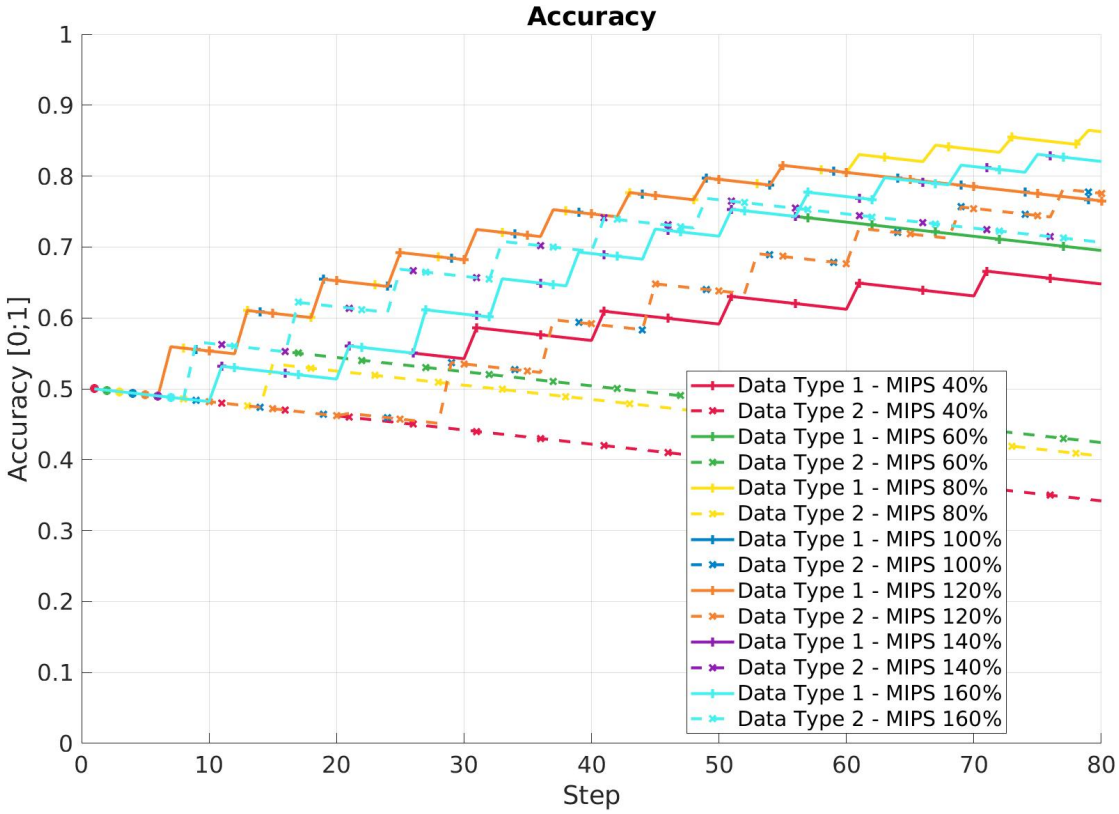
Unusual behavior can equally be observed for scenarios 1, 2, and 3 that have one output completely neglected by the optimizer. It looks like the optimizer favor one output over the other, despite the penalty of having a low accuracy value. This behavior is explained with Figure 5.15b which shows the processing resources left at the PL OBC. It can be clearly noted that they drop to almost zero for the first three scenarios. Moreover, it has to be remembered the second algorithm is more expensive to run in terms of processing resources which could explain the use of the first one.

The ultimate parametric analysis performed in this chapter is about the weights in the objective function. The goal is to compare the result when one or the other data output is put in favor. This represents a case where one type of output is more important than the other for the overall mission success. As for the other analysis, the goal is to understand the effect on the overall accuracy. For this comparison, the weight of the first one is shifted from 0.4 to 1.6 with a step of 0.2. In opposition, the second data output has a weight varying from 1.6 to 0.4 with the same increment. The goal is to emphasize the behavior of the optimizer while the importance between the two outputs is changed.

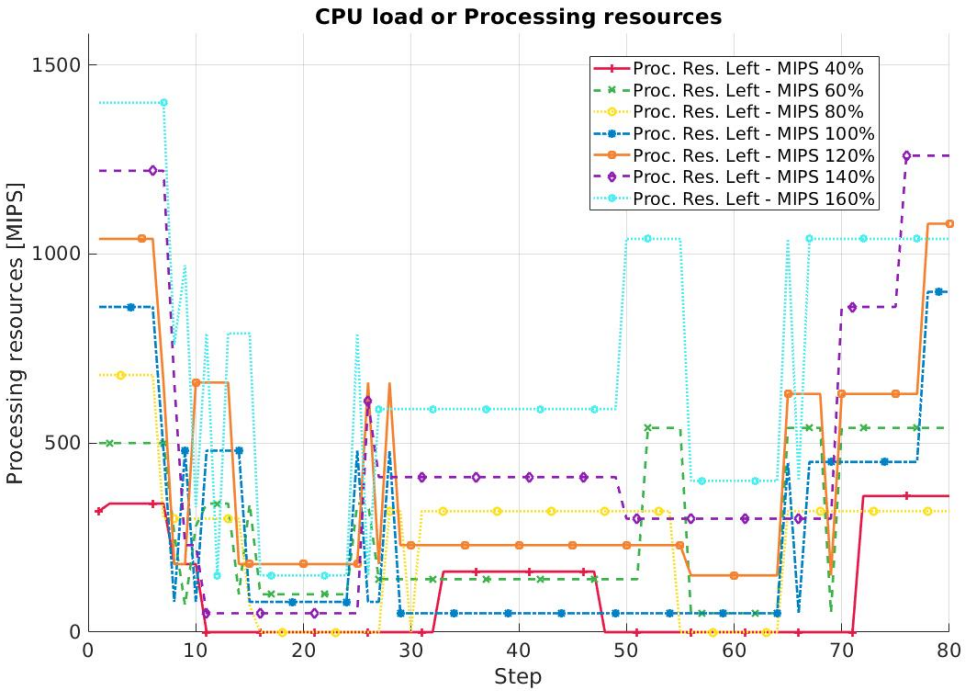
In Figure 5.16a, two different trends can be observed. For the scenarios with reduced weight on the first output, the second one is favored and reversely. The exception is, of course, scenario 4 which is the balance one with both weights equal to 1. In this case, the optimizer switches the algorithm to use between the beginning and the end. This specific case was already analyzed in the previous subsection. The unusual fact is the lack of in-between trends in the optimizer. There are exclusively three cases possible. One solution could be to simulate with a lower increment step to potentially see a difference. By looking at the memory usage in Figure 5.16b, the variation is shown in the various scenario. Nonetheless, the differences between the two extremities can still be distinguished with the memory being used.

4 Comments

For the first parametric analysis, the trends tend to show the tight constraint between the electrical energy and the overall result of the optimizer. By modifying any parameters linked to energy, the accuracy on target undergoes large modification. The low energy phase of the system imposes the strongest constraint on the payload avionic architecture. It is a concerning result since the energy level should in theory not drive the operation of a spacecraft to this level. Such a case would imply a modification of the power subsystem of the spacecraft. Another solution would be to split the task of the system over a different period. Something similar can already be observed in Figure 5.7a where the optimizer alternate between the data type 1 and 2. Nevertheless, this effect should be amplified to better support a true operation flight phase. This alternative is described in greater detail later in the thesis.

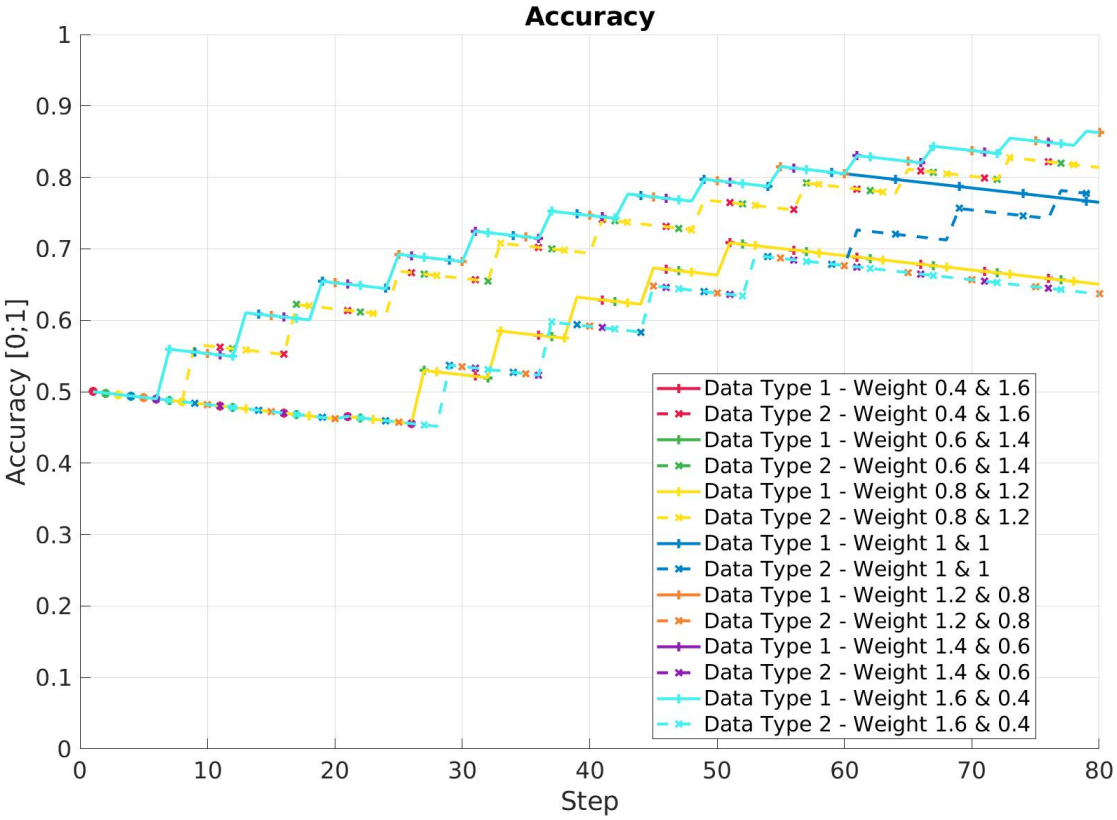


(a) Accuracy.

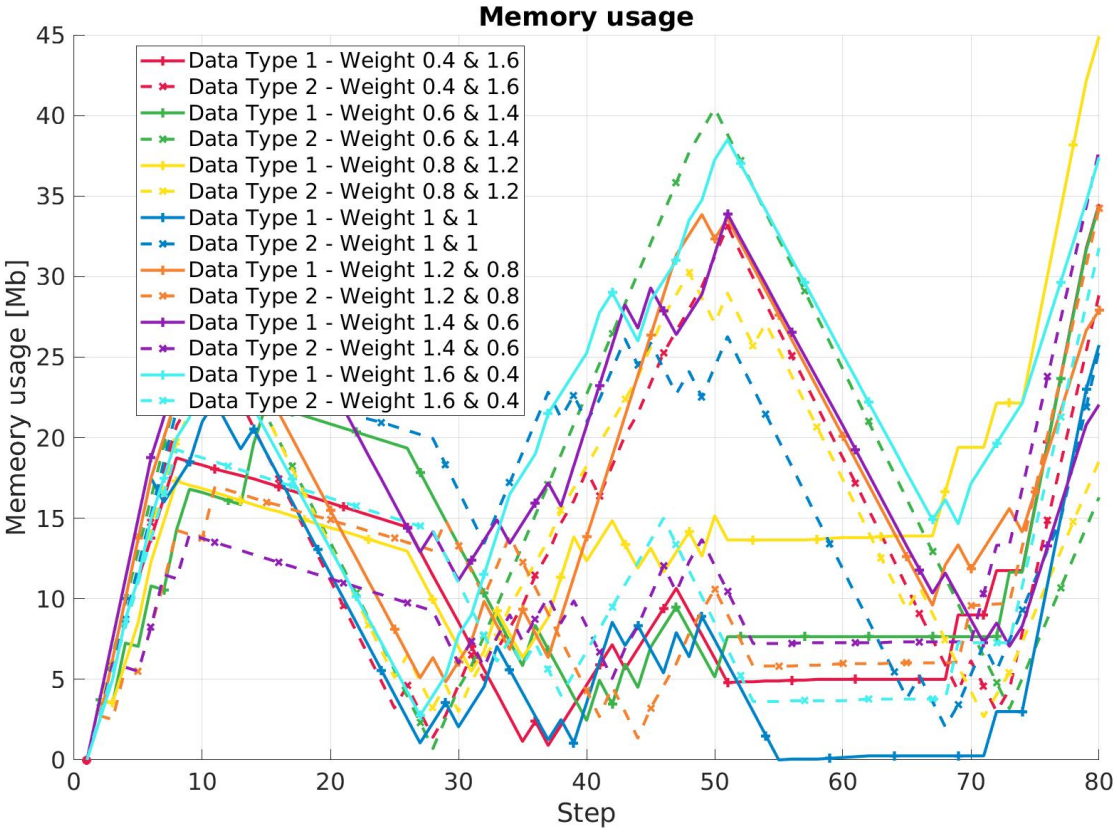


(b) Processing resources left.

Figure 5.15 – **MIPS at PL OBC comparison.**



(a) Accuracy.



(b) Memory usage.

Figure 5.16 – **Weight comparison.**

Regarding the energy, an interesting topic would be to look at the correlation between the period of the electrical energy generation and the ones of the algorithms. In one of the parametric analyses, the energy period was modified, and the result tends to indicate that longer ones were not ideal. Nevertheless, it might be linked to the algorithms' frequencies being similar.

Looking at the other parameters tested, a promising result is the lack of consequent changes as the α parameter is modified. It means it is less essential to design properly such parameters since it does not affect the general behavior of the payload avionic.

Another point would be to examine the result of some simulations with an extended duration. For example, the result with the PL OBC processing resources might suggest considerable differences over time. Because of the brief duration, some behavior might have been missed. Nevertheless, this analysis shows the correlation between this parameter and the overall accuracy result. As expected, the capability of the PL OBC is key to the success of the mission.

Lastly, the weight analyses on the objective function indicates a strong reaction from the optimizer. By changing uniquely the value by a few percent, it is possible to entirely change the operation of the payload avionic architecture. It is a very interesting result since it indicates the sensitivity of the tool to this specific parameter. It means that the user should take special care in designing the weight to obtain the desired output.

On the general approach followed in this chapter, the scenario and architecture tested are not representative of any real case. Even though the goal has been to evaluate the capability of the tool with simple design and constraint, it is fundamental to elaborate a more advanced model. It is assumed that the result shown in this chapter are broadly applicable to any other situation. Nevertheless, a representative model needs to be implemented to test this assumption. It is the topic of the last chapter of the thesis.

5 Outcome & Incoming Challenges

The current version of the optimizer delivered a broad range of improvements that will help to define the optimal payload avionic architecture. The addition of parametric analyses and the more efficient mathematical implementation of the model unlock considerable flexibility to follow the development of the payload avionic platform such as for ClearSpace-1 (CS-1) satellite.

This chapter has proved the capability of the optimizer to perform an extensive range of analyses. One significant advantage in the implementation is the prevention of unfeasible scenarios. In addition, with the reference scenario at the starting point, it is currently possible to compare several aspects of the architecture by varying one or more constraints. This feature enables a deeper understanding of the tight interactions happening in the system. The knowledge acquired by performing parametric analyses will facilitate a cleaner implementation of the

avionic for CS-1. Moreover, it is more straightforward from a mission planning point of view to start with an existing optimal solution to extract a feasible plan.

It is critical to notice the tool is still under development. Many details need to be corrected and some parts will require modification in the future. One decisive point is the explosion in the number of variables as the quantity of iterations increases. Indeed, each new step added to the optimizer needs one extra variable per element in the system. With the current reference architecture, there are 27 variables required per step of the simulation. It means doubling the simulation time implies more than 2000 new variables. This complexity rise causes massive issues with the overall optimizer solving time. Even though the various matrices involved for the constraint are highly sparse, the optimizer requires a large amount of time to converge on a standard computer. One workaround would be to run the optimization on a dedicated powerful machine.

As mentioned in an earlier subsection, the accuracy of the model remains an issue that needs to be addressed. It is right now difficult to validate any outputs of the optimizer since the level of abstraction is relatively high. Naturally, there are possibilities to verify part of the model with a more simple avionic. Indeed, by creating a basic hardware setup, it would be theoretically possible to compare part of the result with the optimizer output. This step represents the focus of the following chapter. Evidently, the tool is designed for initial assessments of payload avionic architectures and thus it makes the validation less crucial. Nevertheless, flexibility needs to be incorporated into the general structure of the optimizer to follow the development of the satellite. If the design choice and the properties of the elements can be integrated as they are defined for the mission, the tool will provide constant feedback on the general payload architecture.

6 Acknowledgments

The authors would like to thank again Prof. Dr.-Ing. Timm Faulwasser from TU Dortmund for helping with the mathematical model and providing multiple feedback on the project.

6 Optimizer verification

1 Introduction

This chapter is based on the publication "Validation of Optimal Control Design Tool for Dedicated Avionic in Active Debris Removal Mission" which was presented in March 2022 at the IEEE Aerospace Conference in Big Sky, MT, USA[155]. The content has been adapted to the structure of the thesis. The challenge presented in this part is to undertake the verification of the optimizer shown in the previous chapters. To achieve it, the primary task is to implement representative software embedded in real hardware.

Mockups of the payload avionic architecture have been implemented in a real processor to compare its behavior with the result of the optimizer. The goal is to create a Hardware-In-the-Loop (HIL) setup where components can be assessed. It allows verifying the mathematical model implemented in the optimizer. By using a low complexity Payload On-Board Computer (PL OBC) with simplified software, some preliminary analyses are conducted. Multiple scenarios are created inside the optimizer, and the results are compared relative to the real hardware. The differences observed are fundamental to understanding the flaw and limits of the optimizer tool. They also emphasize the value of certain types of solutions and the achievability of produced designs. The purpose of this chapter is to compare the behavior of a HIL setup with the optimizer result by using a basic payload avionic architecture. To simplify the process, first-hand approximations are used for the parameters of the payload, and the scenarios are kept at a low complexity level.

2 Verification Process

To simplify the first step of verification, it has been decided to limit the investigation to the algorithms and the processors part. As presented in the two previous chapters, the optimizer includes, in addition, memory and external sensors. These components might represent the work of a forthcoming study. In summary, the goal of this first verification is to compare the result of the optimizer presented previously with algorithms running in true hardware. To

achieve this, mockup algorithms have been created and implemented in a low-level fashion. The primary purpose is to have control over the number of operations done inside each of them. These algorithms are then compiled into a limited program and run with some constraints by a master script. Every step of the script is being monitored by some external tool and the output of the mockup algorithms is stored in files. Ultimately, the results are analyzed and compared relatively to the optimizer output.

2.1 Algorithms Definition

As said in the previous part, the first step is the definition and implementation of the algorithms. As a reminder, these algorithms can represent any process inside the PL OBC relative to an ADR purposes. Typical examples are image processing or GNC algorithms. To decouple the verification from any specific algorithm implementation, some low-level mockups have been designed. The primary constraint is to maintain maximum flexibility in their behavior while keeping them as simple as possible. In addition, it should be possible to extract the exact number of operations done by each algorithm and modify their behavior easily. In other terms, they should be close to machine code and therefore include a minimum usage of external libraries. Indeed, libraries create external dependencies with unknown behavior. To satisfy these various constraints, the algorithms are coded using Assembly language (ASM). The language has the advantage of being low-level and controlling directly the behavior of a computer. This decision is not linked to any future real algorithm implementation. The primary drawback remains its direct link with the hardware being used. Moreover, such an implementation is not representative of real algorithm implementation. Because of the way it is written, the various instructions are completely dependent on the processor of the machine.

To retain the level of complexity down, the primary target is to write code easily readable and modifiable. It is not envisioned to implement any real algorithms at this point since they are overly dependent on the application of the mission. Nevertheless, having real algorithms incorporated in the verification testbench would be greatly beneficial to the overall study in terms of representative behavior.

To approach the implementation aspect, the code is formulated to execute a series of mathematical and storage operations. It assumed that most algorithms would be primarily composed of these two types of instruction. To reduce the number of lines that would effectively need to be written, multiple levels of loops inside the Assembly language code have been implemented. The main control in term of operations quantity is achieved by defining the number of loops that is happening inside each algorithm. The standard way is to first do a few operations and have a first inner loop around it. It establishes the base for the works of the algorithm. After that, the script is forced to perform a pause or break before having a second loop that repeats the whole process. Even though this method adds multiple lines, it allows a finer and more effective way of controlling the algorithms. These mockups' implementations already allow the verification of the mathematical model inside the optimizer without incorporating any

```
39  _loop:
40
41  ;Store some constant
42  MOV R1, #332
43  MOV R2, #448
44
45  ; do 4 multiplication
46  MUL R0, R1, R2 ; the multiplication
47  LDR R6, addr_nb1 ;load address
48  STR R0, [R6] ;store value in memory
49
50  MOV R2, #123
51  MUL R0, R1, R2 ; the multiplication
52  LDR R6, addr_nb2  ;load address
53  STR R0, [R6]  ;store value in memory
54
55  MOV R2, #400
56  MUL R0, R1, R2 ; the multiplication
57  LDR R6, addr_nb3 ;load address
58  STR R0, [R6] ;store value in memory
59
60  MOV R2, #234
61  MUL R0, R1, R2 ; the multiplication
62  LDR R6, addr_nb4 ;load address
63  STR R0, [R6] ;store value in memory
64
65  ;1st loop
66  ADD R3, R3, #1 ; to count
67  CMP R3, #4096
68  BLE _loop
69
70  BL my_sleep ;sleep for 130us
71
72  ; second loop
73  MOV R3, #0 ; reset counter
74  ADD R4, R4, #1 ; to count
75  CMP R4, #87040
76  BLE _loop
77
78  BL _time
79  LDR R1, [R9]
80  BL write ;write to file id algo(id & mode) + time
81
82  MOV R4, #0 :reset counter 2
83
84  B _loop ;infinity loop
```

Figure 6.1 – **Example of ASM code for a mockup algorithm.**

real algorithm.

An example of the code is presented in Figure 6.1. The first task is to load some values into different registers. These numbers are employed later to perform operations, however, the register's values themselves are unimportant. The goal is to load them with multiple digit numbers to simulate some actual mathematical process done inside an algorithm. The following part is essentially manipulating these values and performing some operations. For this code, there are simply four multiplications performed in succession. As mentioned earlier, the results of these instructions are not crucial. The exclusive goal is to emulate the load of an algorithm. To better represent the behavior, the results are stored inside the external memory of the processor. It could be debated that not each outcome would be stored outside in a concrete case and most operation results would stay in the register part or the RAM. At the end of this bloc, there is the first inner loop implemented. The logic behind this procedure is solely to increment a register until a certain value is reached while doing all the four operations before. When the algorithm arrives at the maximum set, then it does not loopback. Afterward, the algorithm is instructed to pause for a brief amount of time, in this case 130 [μs]. This pause allows managing the number of processing resources used during the test by setting some LE time. Once the break is completed, the first incremented register is reset and the program moves with the following instruction. In this case, it is a second loop to primarily manage the time between each update of the algorithm. The concluding lines are here to mimic its output. In this case, the string written to the console is the current system time, the id, and the mode of the algorithm. It represents the successful runtime of a specific algorithm producing

its desired results such as the accuracy parameters presented in the previous chapters. This information is utilized to reconstruct the behavior of the system after the execution of the verification run is finished. The ultimate step is then to start the whole program again with an infinite loop.

2.2 Tasks Control & Execution Script

At a higher level, the task is to run these various algorithms inside a real processor and monitor them. Multiple choices have to be made to balance the realism and the complexity of the implementation. To simplify the operation, a general bash script is generated from the result of the optimizer. The script dictates the algorithms that need to run, the duration, and also the processor usage. The code has the ability to limit the processor usage of the running algorithms. This limitation would potentially come from the electrical energy fluctuation of the system, nevertheless, it is a more straightforward approach to purely handle a native function instead. For these sets of verification, a Linux library is operated that allows to group multiple processes (or algorithms) together and then applies the processor usage limitation. With this approach, the main script is able to simulate the variation in processing resources with a unique function. In addition, it is required to restrain the selection of the CPU core. With the current processor employed, four cores are available, and it assigns the processes to be split among them to reduce the load. To prevent this behavior, another Linux function is used to force all the algorithms to run on the same CPU core. The usage of multiple ones is, without doubt, interesting but it is adding complexity to the verification.

Another significant part of the bash script is to set the monitoring of the algorithm as well as storing their output products. The first thing performed after launching an algorithm is to redirect its product to a dedicated file. Since they mimic their behavior by outputting string every time the second loop is finished, the task of the general script is to store this information in files. To finish the initialization of the process, an external tool is employed to monitor the resource usage as well as the memory. This tool is practical to track the evolution of the algorithm over time. The output result is employed to analyze the effective processor usage of the system.

Regarding the bash script itself, it is generated based on the result of the optimizer presented in earlier chapters. It initially reads the mode sequence of algorithms and the Payload On-Board Computer (PL OBC) and then generates a timeline. This timeline contains the mode change as well as their timing. Once it is produced, a Matlab code will generate the corresponding bash script to follow the same instruction as the optimizer recommended. The created file includes timing instruction and mode change for all the algorithms and the .

The last point worth mentioning is the usage of the sleep function in the bash script. The primary purpose is to emulate phases where nothing is changing in the control of the process. The algorithms are still running and consuming processing resources, but no control needs to be done.

```
echo "Change CPU limit : 56 %"
sudo cgset -r cpu.cfs_quota_us=560000 cpulimit

sleep 4
echo "Sleep : 4s"

sudo pkill -P $pid1_m

echo "Start Algo 1 mode 4"
sudo stdbuf -o L -e L ./../ASMscript/script_1_4 >> ../dataOutput/out_1.txt
2>&1 & export pid1_m=$!
sleep 0.5 #need to wait a bit
pid1=`pgrep -P $pid1_m`
sudo taskset -apc 0 $pid1 &
sudo cgclassify -g cpu:cpulimit $pid1
time=$(date +"%s")
psrecord $pid1 --log ../ressOutput/activity_${time}_1.txt --include-children
 &

sudo pkill -P $pid2_m

echo "Start Algo 2 mode 4"
sudo stdbuf -o L -e L ./../ASMscript/script_2_4 >> ../dataOutput/out_2.txt
2>&1 & export pid2_m=$!
sleep 0.5 #need to wait a bit
pid2=`pgrep -P $pid2_m`
sudo taskset -apc 0 $pid2 &
sudo cgclassify -g cpu:cpulimit $pid2
time=$(date +"%s")
psrecord $pid2 --log ../ressOutput/activity_${time}_2.txt --include-children
 &

echo "Change CPU limit : 100 %"
sudo cgset -r cpu.cfs_quota_us=1000000 cpulimit

sleep 50
echo "Sleep : 50s"

echo "Change CPU limit : 56 %"
sudo cgset -r cpu.cfs_quota_us=560000 cpulimit
```

Figure 6.2 – **Example of a main script in bash.**

A typical bash script can be seen in Figure 6.2. The various timing and operations are directly linked to the output of the optimizer for a specific scenario. In this part of the script, the first task is to change the available processing resources of the Payload On-Board Computer (PL OBC) (Raspberry Pi). In this case, it is 56% of the maximum available. The following step is solely to let the algorithm run for 4 $[s]$. Afterward, the main script stops the previous algorithm (id 1) running and launches it in mode 4. During the launch, the script first defines where the output is stored, then sets it to continue on core 1 with $taskset$. The subsequent step is to activate the CPU constraint on that and finally start the monitoring with $psrecord$. The bash script performs the identical procedure with algorithm id 2 and then changes the overall CPU usage to 100%. Ultimately, everything runs as planned during 50 $[s]$ before the CPU limit is set back to 56%. More information regarding the mode of the various elements is provided in the following sections.

2.3 Results Extraction

Once the set of processes is executed, it is possible to reconstruct the entire test. Real-time observation of the system is possible, but the comparison with the optimizer is not ideal. For this reason, all the results of the processes are collected in various files that can be read later on. These files contain information about the algorithms' updates and their current mode. There are, furthermore, the recordings of the processing resources (or CPU mode) and memory usage of the tasks.

All information is imported into another Matlab script that will extract the needed details to reconstruct the scenario. It will also compare the various timings from both the recording tools and the algorithm to ensure synchronization. To compare these results with the optimizer output, some definitions of the timestep are required. It can be set arbitrarily to match the various parameters of the optimizer. Comparisons plots will be shown in the following section of this chapter.

2.4 Hardware

The concluding part is the description of the hardware used. For simplicity, a Raspberry Pi 4 Model B (IC: 20953-RPI4B)[157] has been chosen. It offers considerable flexibility and an impressive range of Input/Output (I/O) ports. This model has Quad-core Cortex-A72 (ARM v8) 64-bit System-on-Chip with a clock frequency of 1.5 $[GHz]$ and 8 $[GB]$ of SDRAM. It runs on the Raspberry PI OS based on a Debian architecture. As mentioned earlier, the current verification process uniquely uses one of the cores for the algorithms. It allows employing the other ones to handle the recording and monitoring tools. This specific hardware has been used mainly as a convenient way to test embedded software. This electronic board is not representative of any real PL OBC but it allows for a cheap and efficient way to verify the mathematical model of the optimizer.

3 Verification Results

The purpose of this section is to compare the results obtains from the optimizer with the ones of the hardware setup.

3.1 Optimization Scenarios

For the first type of verification, uniquely one payload avionic architecture is implemented. The idea is to establish a baseline to verify the various function and outputs of the optimizer. As mentioned earlier, the optimizer is first run with scenarios, and the output is used to generate the verification code. It means the same payload architecture with identical constraints is implemented in the hardware. In this case, the classic payload architecture from the previous chapter is used with two sensors, two memories, and two algorithms. The Figure 6.3 shows its layout. The key point is a single sensor connected to a unique algorithm through a memory. It is preferred to avoid correlation between the two sensors especially when they are untested in the hardware part. The primary objective is to evaluate the processor and its two algorithms. The description of the parameter for each mode can be found in the previous chapter with Figure 5.4 & Figure 5.5. This chapter uses identical characteristics for both the PL OBC and the algorithms.

To investigate it properly, four sets of two scenarios have been run in the optimizer with significant differences in their results. The goal is to look at a variety of cases to better understand the differences and similarities with the hardware setup. To obtain a more diverse set, scenarios from the previous chapter are taken. The selection is based on multiple parametric diversifications, and extremes in the result are chosen. Nevertheless, each individual optimization was already shown in the previous chapter inside one of the parametric analyses. In the following part, the verification is performed on the change of the electrical energy amplitude, offset, and period as well as the PL OBC CPU load (or processing resources) variation. As a reminder, these scenarios are not representative of any flight phase for an ADR satellite. They are designed

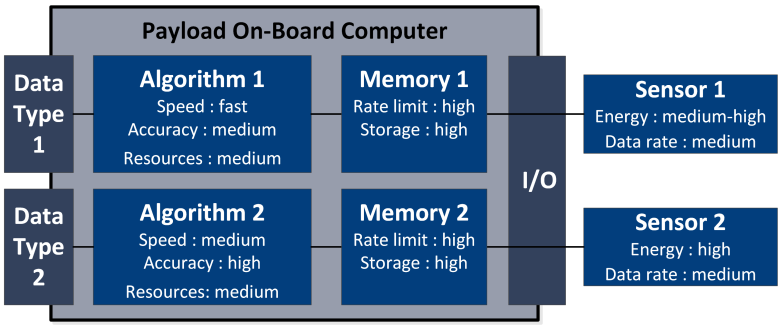


Figure 6.3 – **Architecture of the payload avionic used.**

to put specific constraints into the optimizer. Regarding the reaction of the payload avionic, it is important to remember the task of the optimizer. For the scenario presented below, the tool has tried to optimize the mode of each element at every time step to obtain the highest accuracy per data type. The user of the tool has no control over this mode variation and can uniquely change the general constraint of the optimization through the scenario design.

- The first scenario implies the amplitude of the electrical energy being altered. It was decided in earlier work to assess the effect of sinusoidal electrical energy generation to represent a slowly spinning satellite. The hypothetical spacecraft would incorporate solar panels with varying orientations toward the sun and thus variation in the electrical energy generation. In addition, it is assumed that the satellite includes batteries to provide a minimum level. For this scenario, the variation due to the spinning axis orientation is set to respectively 40 $[W \cdot h]$ and 80 $[W \cdot h]$ amplitude. This difference could also reflect two distinct types of solar cells used in the satellite.

- The second scenario focuses on the electrical energy offset. In this case, it can be identified as a trade-off between two different sets of batteries incorporated in the satellite. In the first case, the batteries would provide a minimum of 100 $[W \cdot h]$. For the second, it is assumed to deliver 220 $[W \cdot h]$.

- The third scenario looks at the periodic fluctuation in electrical energy. The logic behind this variation would represent the spin rate of the satellite. By adjusting the rate, the solar panels would be exposed to the sun for less time but more often. This rotational rate could be linked to the alignment of the chaser to the target state. In this analysis, the satellite is doing one rotation in 20 $[s]$ and 120 $[s]$.

- The final scenario is regarding the CPU load on the Payload On-Board Computer (PL OBC). It is related to the type of computer used and the capabilities of the processor. The first case has merely 40% of the nominal processing resources (regarding the base scenario) and the second one has 160%. This percentage is affecting the available resources in every mode of the PL OBC.

For all the scenarios presented in this chapter, the optimization and thus the verification will last for 160 $[s]$. It means it simulated the constraint on a payload avionic architecture for this amount of time. It is, of course, a relatively short-period if compared to an orbit in LEO. Nevertheless, the goal is to assess quickly the reaction of the system in the HIL setup and verify that it matches the mathematical model of the optimizer. At this point, it is not to test representative scenarios.

3.2 Comparison of Behaviors

Each of the following points discusses the results for one set of two scenarios.

Electrical Energy Amplitude

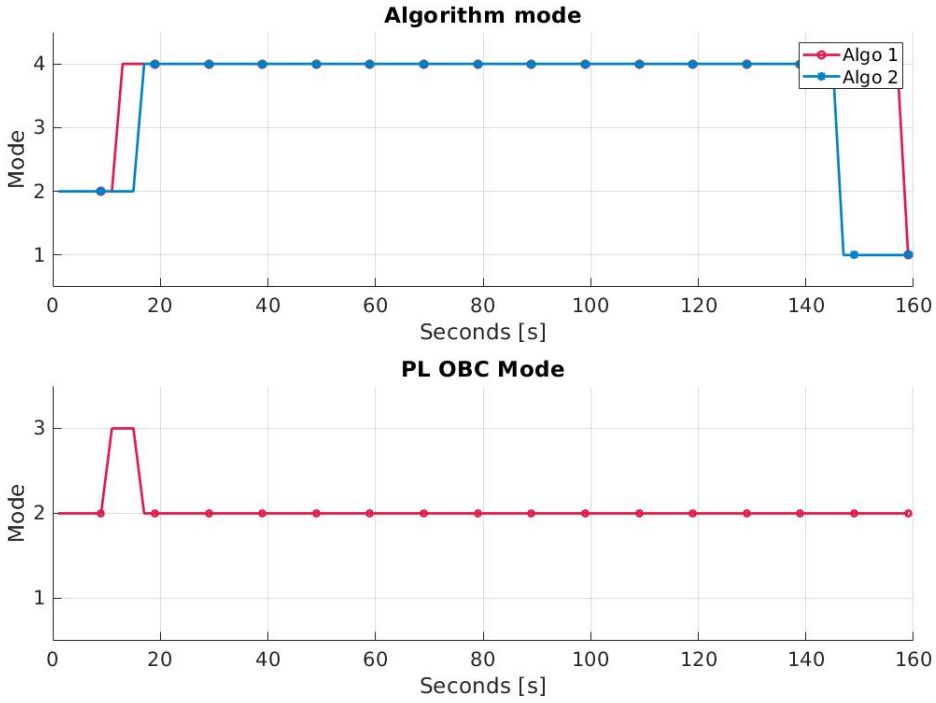


Figure 6.4 – **Electrical energy amplitude** 40 $[W \cdot h]$ **- Modes over time.**

As mentioned earlier, this scenario looks at the variation in the amplitude of the electrical energy generation. Figures 6.4 & 6.7 show the mode variation for both the PL OBC and the algorithms in the two cases. In the first scenario, the algorithms stay in state 4 (the most powerful one) for almost the entire test with the exception of the beginning and the end. The PL OBC is also staying in mode 2 (the most powerful one) for almost the full duration. In the second scenario, the first algorithm varies from state 4 to state 1 (turn-off) later on. The second one stays in mode 2 and then moves to mode 4. The idea behind this is to keep both accuracy variables at a high level when there are not enough processing resources for the

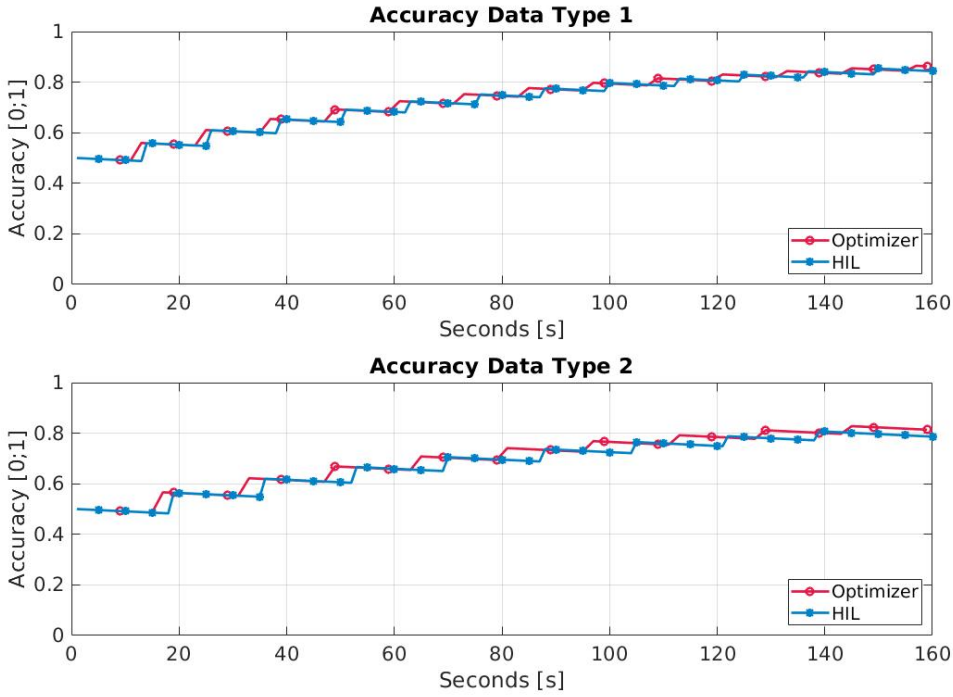


Figure 6.5 – **Electrical energy amplitude** 40 $[W \cdot h]$ **- Accuracy per data type.**

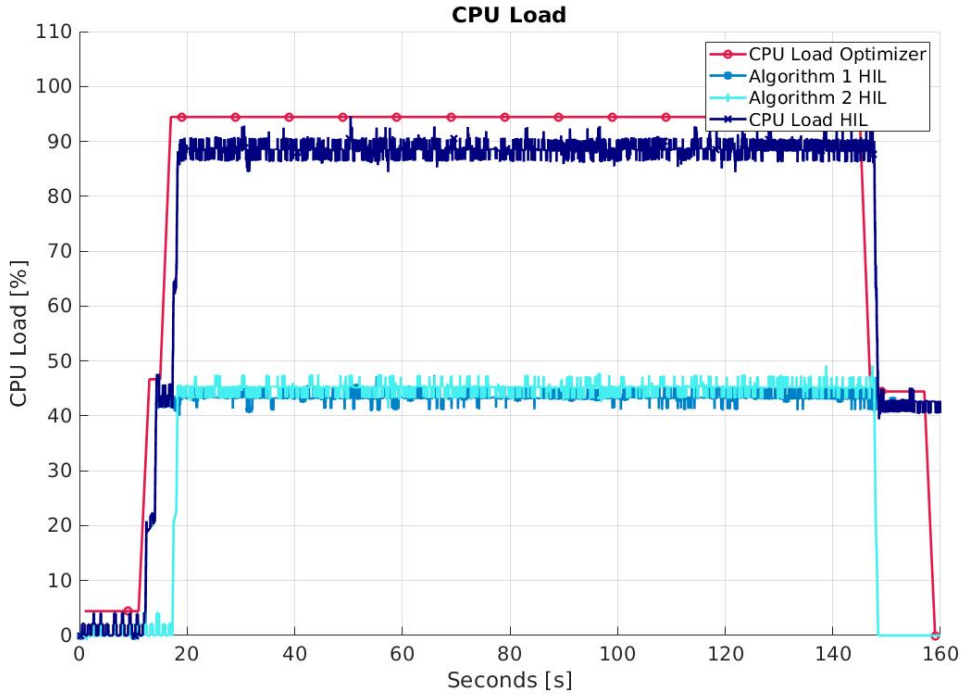


Figure 6.6 – **Electrical energy amplitude** 40 $[W \cdot h]$ **- CPU load.**

two algorithms. Regarding the PL OBC in the second case, fluctuations between state 2 and 3 happen when the electrical energy is low. The long period in state 2 allows both algorithms to be in mode 4.

Looking at Figure 6.5, the regular trend of the algorithms can be observed for both the optimizer result and the Hardware-In-the-Loop (HIL) output. On average, the end results in both cases are in the order of a few percent. Nevertheless, the HIL setup looks to suffer a little delay in the update of the algorithm. This specific fact will be discussed later, but one of the reasons could potentially be the timing synchronization of the two results. To be noted that the HIL setup receives typically a lower number of updates. For both algorithms, the difference is 1 update. The reason comes from the actualization frequency of the mock-up algorithm.

Figure 6.6 shows the variation in CPU load or processing resources used in the optimizer as well as in the HIL setup. The plot additionally includes the resources used by both algorithms separately. In these types of graphics, the results of the HIL have been placed first through a smooth function to eliminate the noise. Indeed, because of the nature of the recording, many significant variations are monitored during the testing. The smooth function (a simple moving average) allows an easier display of the general trend. Regarding the results themselves, the matching with the timing is equally good. The general trend of the HIL setup accurately follows the optimizer result. The discrepancies are discovered in the number of resources used. The HIL algorithms demonstrate the tendency to consume less than what is expected. This behavior could be linked to the algorithms' implementation. The phenomenon is discussed later in the chapter.

Figure 6.8 displays a similar result than Figure 6.5. The key difference is that the number of updates for the accuracy 1 is identical for the HIL environment and the optimizer. Regarding the second algorithm, the HIL setup is missing its last update (6 in the HIL versus 7 in the optimizer) and thus the finished result is marginally different. In both cases, the difference can be explained by a mismatch in the timing synchronization between the optimizer and the setup. It happens when the HIL software takes longer to initialize than the optimizer simulation. In addition, the timing of the algorithm update is moderately longer in the HIL environment.

Regarding Figure 6.9, the overall comparison is promising. The general trend of consumption is similar with few mismatches. The total usage is marginally reduced for all parts except for the last one of the tests regarding the setup. It also looks like the middle and the final phase are somehow shifted to the right. The assumption hither is once more some discrepancies in the timing synchronization of the two tests.

Electrical Energy Offset

Figures 6.10 & 6.13 show two complete opposites in term of behavior for the mode. In the first scenario, the electrical energy available is inadequate. It means the PL OBC has to stay in

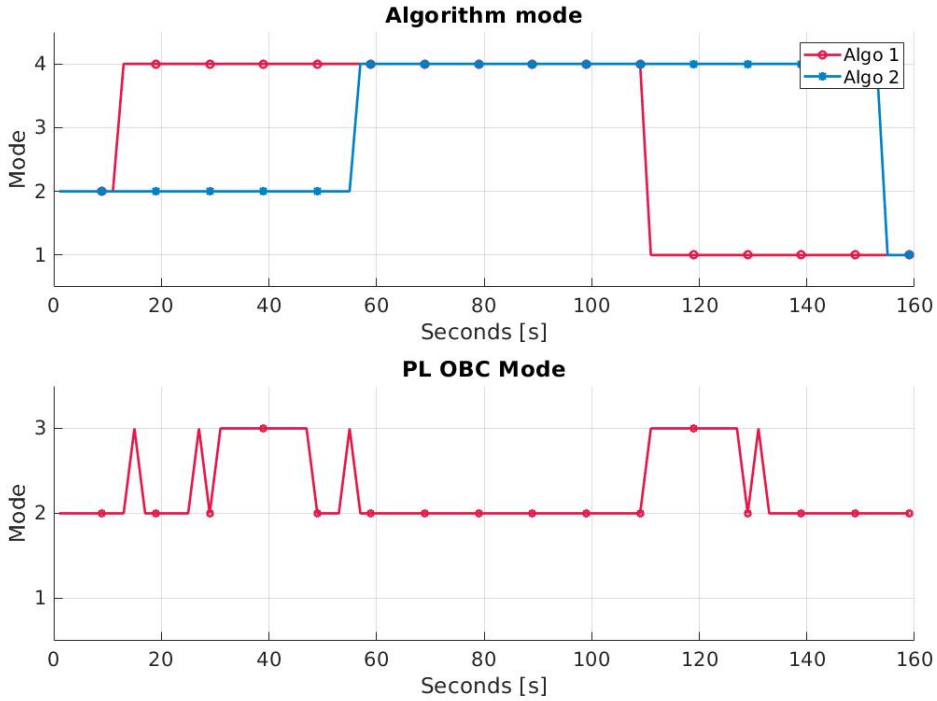


Figure 6.7 – **Electrical energy amplitude** 80 $[W \cdot h]$ **- Modes over time.**

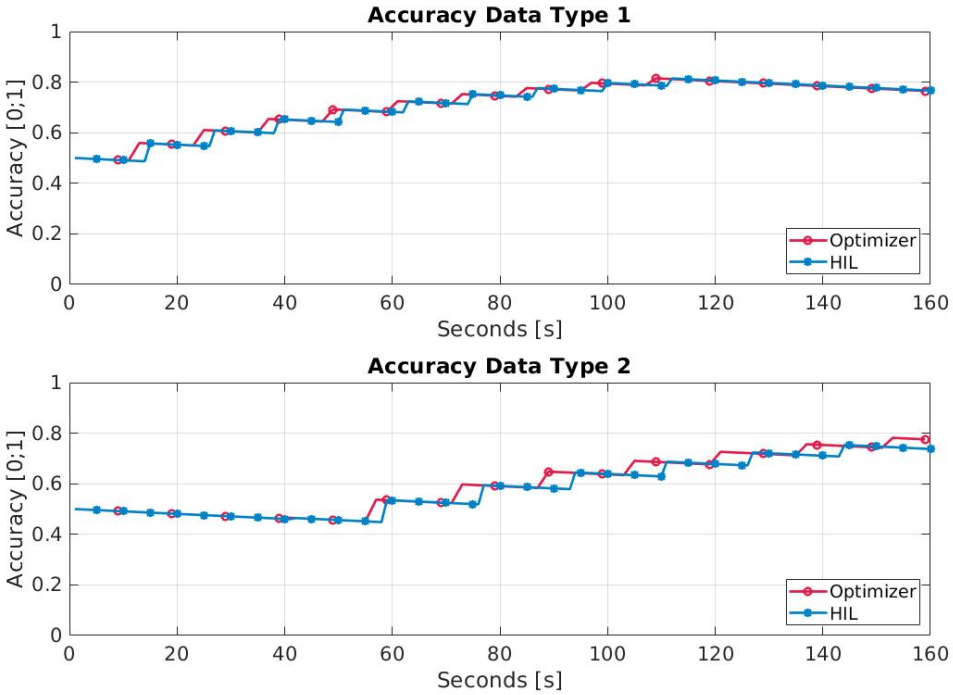


Figure 6.8 – **Electrical energy amplitude** 80 $[W \cdot h]$ **- Accuracy per data type.**

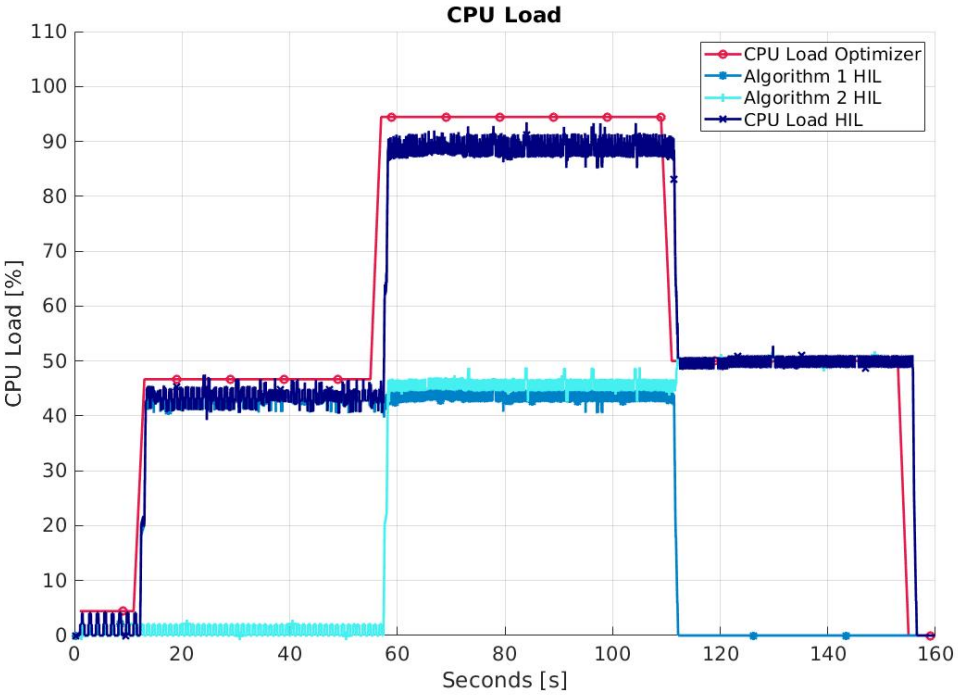


Figure 6.9 – **Electrical energy amplitude** 80 $[W \cdot h]$ **- CPU load.**

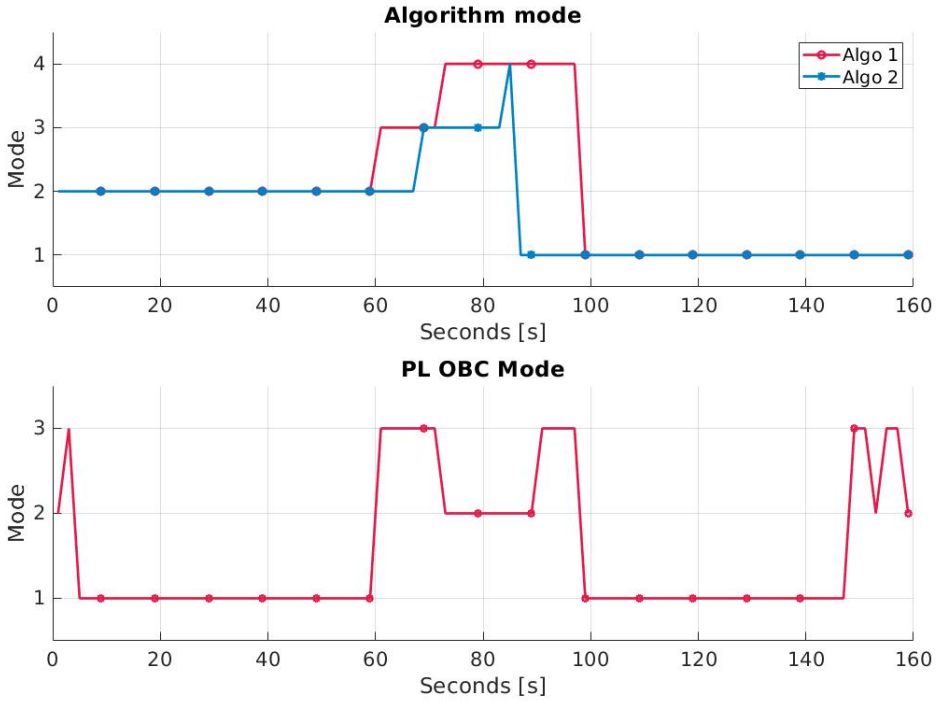


Figure 6.10 – **Electrical energy offset** 100 $[W \cdot h]$ **- Modes over time.**

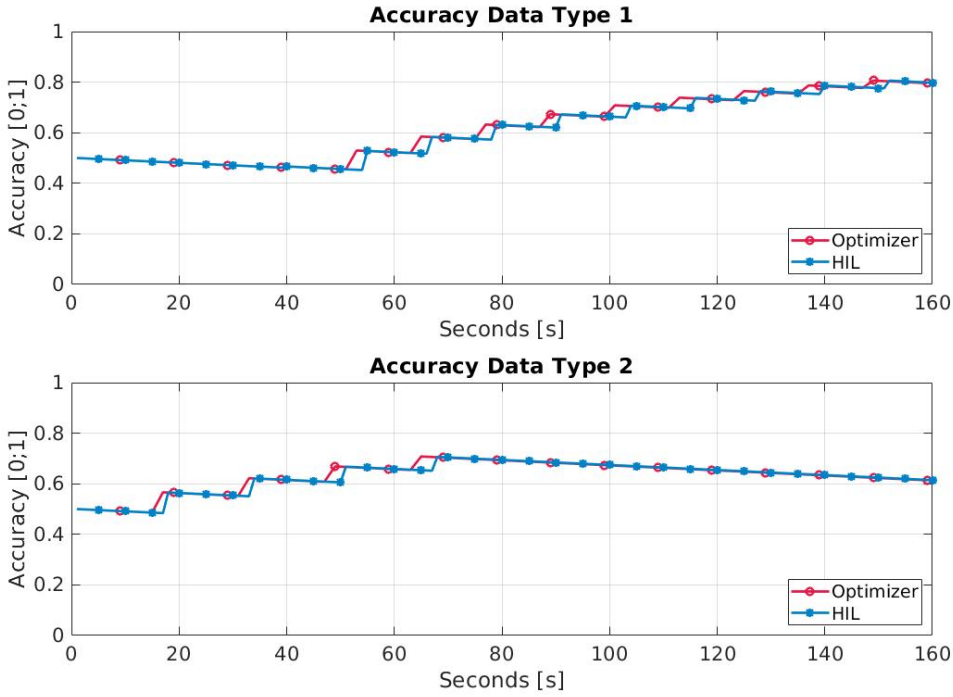


Figure 6.11 – **Electrical energy offset** 100 $[W \cdot h]$ **- Accuracy per data type.**

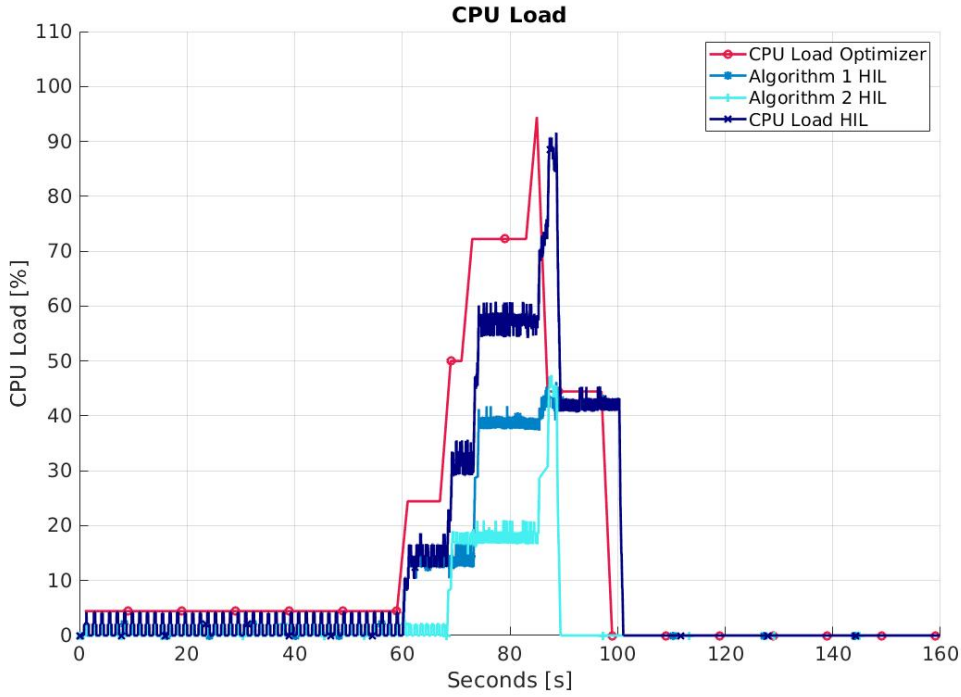


Figure 6.12 – **Electrical energy offset** 100 $[W \cdot h]$ **- CPU load.**

the Low Energy Consumption (LE) consumption state (mode 1) most of the time with some exceptions in the middle phase and at the end. There are merely a few seconds when it can be in full mode (mode 2). As consequence, both algorithms have to be in the LE state (mode 2) or turn-off (mod 1) during the low phase. In the middle phase, uniquely algorithm 1 is able to be run in full mode. In comparison, the PL OBC in Figure 6.13 (second scenario) can stay in full mode during the whole test. By consequence, the algorithms are set to their powerful state as well during almost the entire test. Regarding the first scenario, it is obviously not a desired behavior to run out of electrical energy from a mission planning point of view. Nevertheless, the goal of these tests is to verify the mathematical model of the optimizer, thus this behavior is not critical.

Figure 6.11 demonstrates the evolution of the accuracy during the test. Since both algorithms are running exclusively during a short period of time, the number of updates between the HIL setup and the optimizer is the same. Nonetheless, the timing is slightly shifted over time, and a delay appears.

Figure 6.12 does not look ideal on first sight. The general trend is similar, but the timing shift is higher than in other scenarios. Moreover, the optimizer is showing a higher CPU load over almost the entire intermediate phase. In the worst case, the difference reaches up to 15%. In this situation, it is probable the actual processor of the HIL setup is experiencing some additional constraints. The intriguing result happens during the peak around 90 $[s]$ of the test where the maximum is almost rising to a similar level for both curves. Even though the output is shifted in time, the peaks are virtually identical.

For the second case, Figure 6.14 shows both algorithms being updated multiple times. For the two data types, the HIL setup is missing one final update to be at the same level as the optimizer. It is fascinating to notice that the accuracy data type 1 looks less affected by this mismatch than the other one. When the update is high, it is by default experiencing less increase. For this reason, the difference looks subtler. Nevertheless, the issue is once more linked to the frequency update mismatch and the timing synchronization at the beginning.

Finally, Figure 6.15 is almost identical as Figure 6.6. In both cases, the general trend is the same for the two tests with again minor differences in the usage level. The differences in the total CPU load look to be in the order of a few percent. Nevertheless, the timing synchronization of the two tests seems better in this case with slight variations.

Electrical Energy Period

Figures 6.16 & 6.19 represent the mode over time for both cases when the electrical energy period is varying. In the first instance, the PL OBC is first set to LE before a time where it alternates between mode 2 and 3. In this case, it is following the period of electrical energy generation. On a side note, this odd behavior is directly coming from the optimizer and more investigation should be done to understand the logic behind it. This might be linked to the

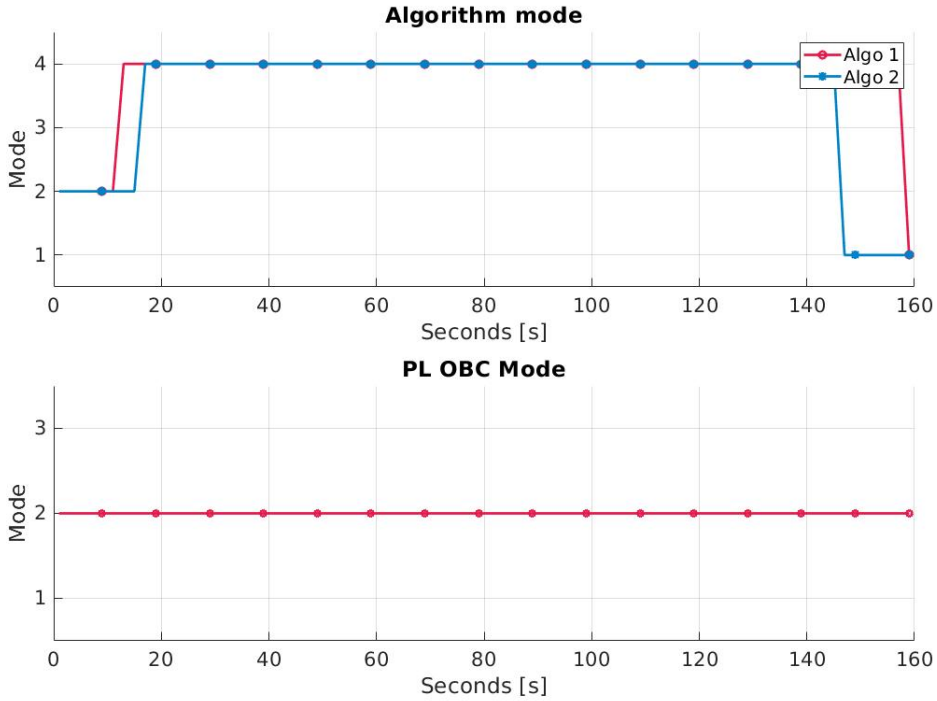


Figure 6.13 – **Electrical energy offset** 220 $[W \cdot h]$ **- Modes over time.**

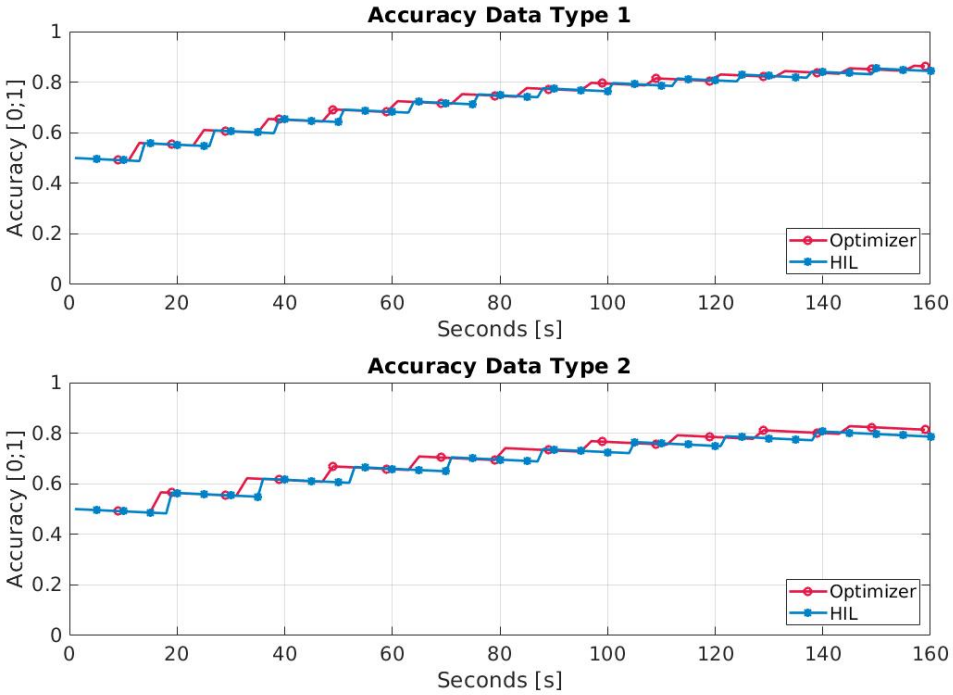


Figure 6.14 – **Electrical energy offset** 220 $[W \cdot h]$ **- Accuracy per data type.**

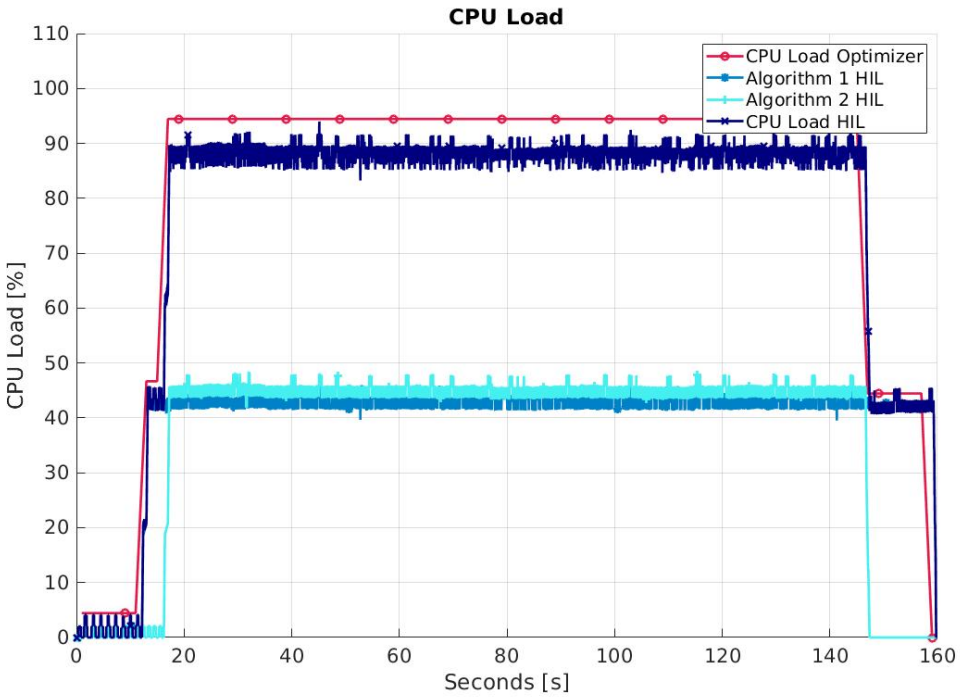


Figure 6.15 – **Electrical energy offset** 220 $[W \cdot h]$ **- CPU load.**

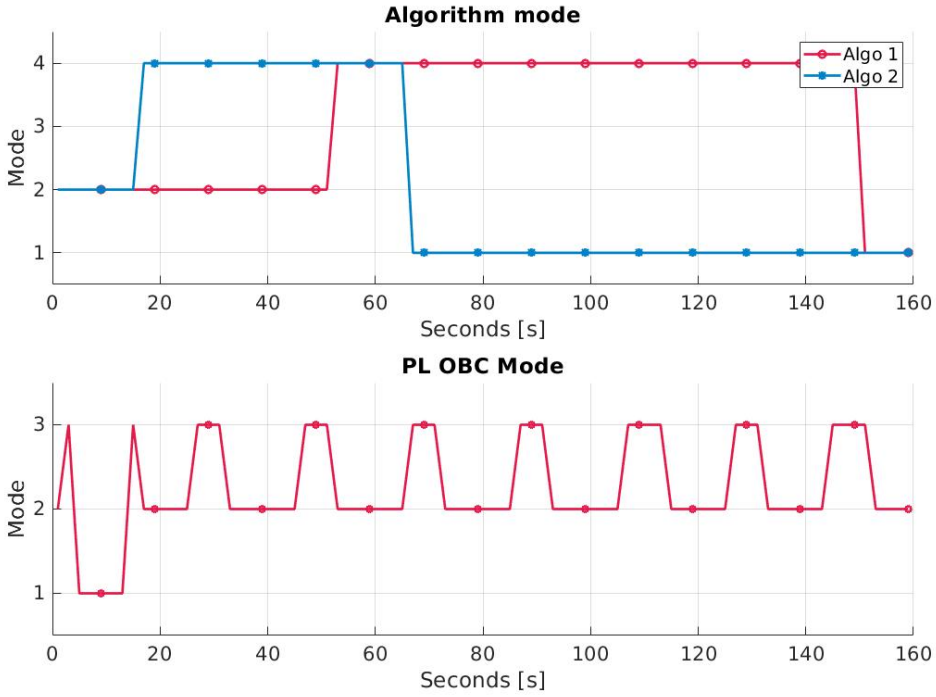


Figure 6.16 – **Electrical energy period** 20 $[s]$ **- Modes over time.**

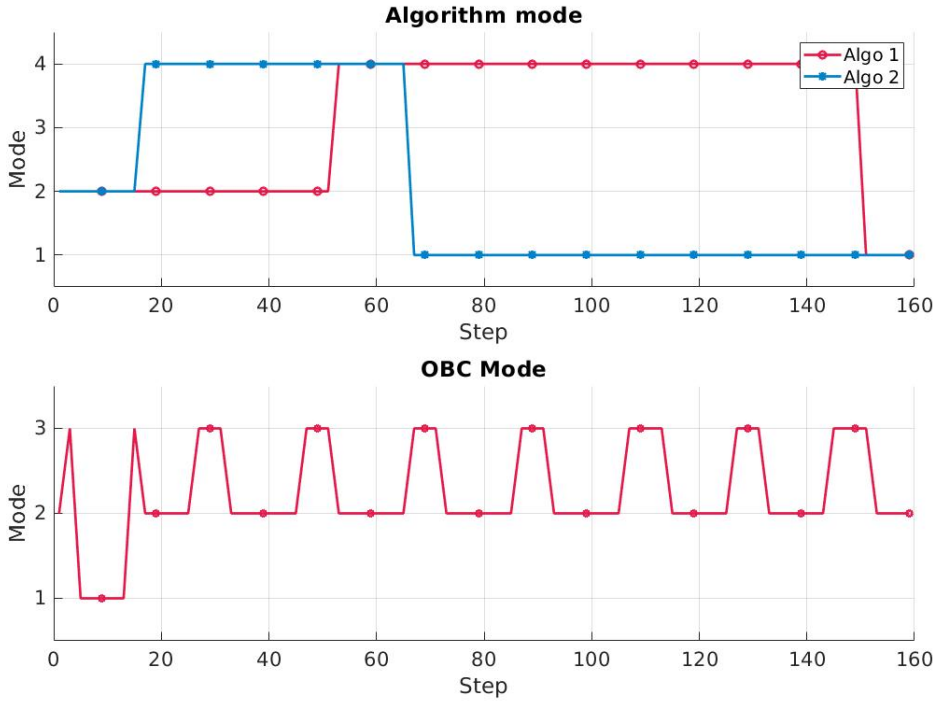


Figure 6.17 – **Electrical energy period** 20 [*s*] **- Accuracy per data type.**

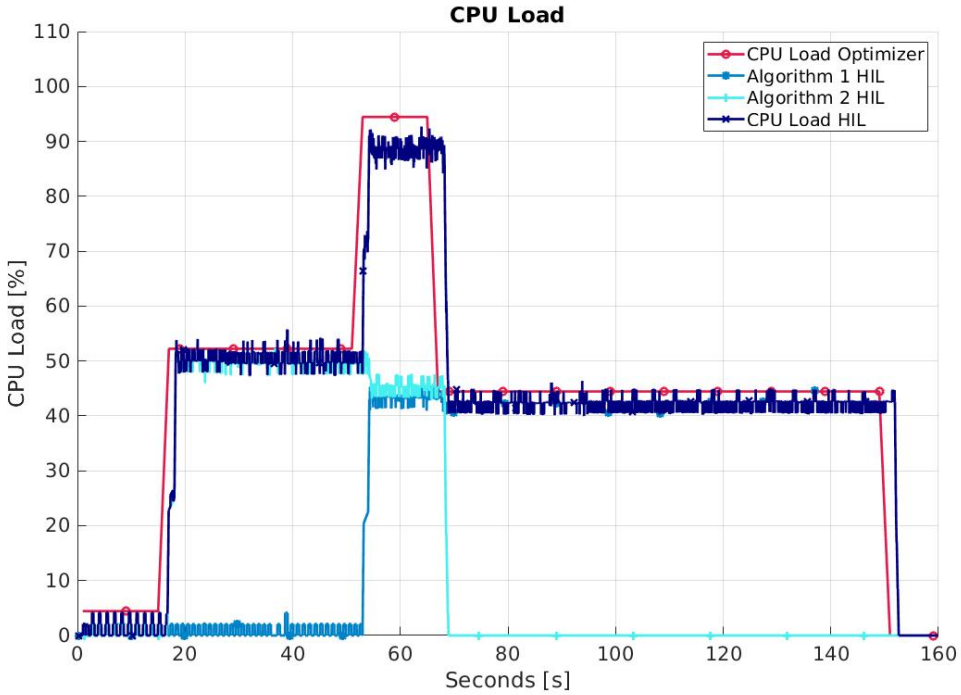


Figure 6.18 – **Electrical energy period** 20 [*s*] **- CPU load.**

mode of the sensors which are not shown here. Because of this behavior, the algorithms can not be operated in full mode at the same time. It results in the optimizer choosing the second one and then the first one. In the second case, the PL OBC is slightly more consistent in modes with two extended periods in full state and some in-between fluctuation. The mode variation in the middle might again be linked to the sensors' mode. The result is that algorithm 2 is in full mode for a short period. In comparison, algorithm 1 stays at maximum efficiency during almost the entire test.

Figure 6.17 shows that no mismatches are happening in the number of updates. Nonetheless, a similar behavior as for the other scenarios is observed. The HIL setup experiences some delay in the algorithm's frequency and is subject to a slight timing de-synchronization. At the end of the test, the results for both data types are identical between the optimizer and the setup.

Figure 6.18 suffers similar issues as for previous scenarios. The total CPU load consumed is lower in the setup and has some delay. Nevertheless, the general trends are matching and the differences are minimal.

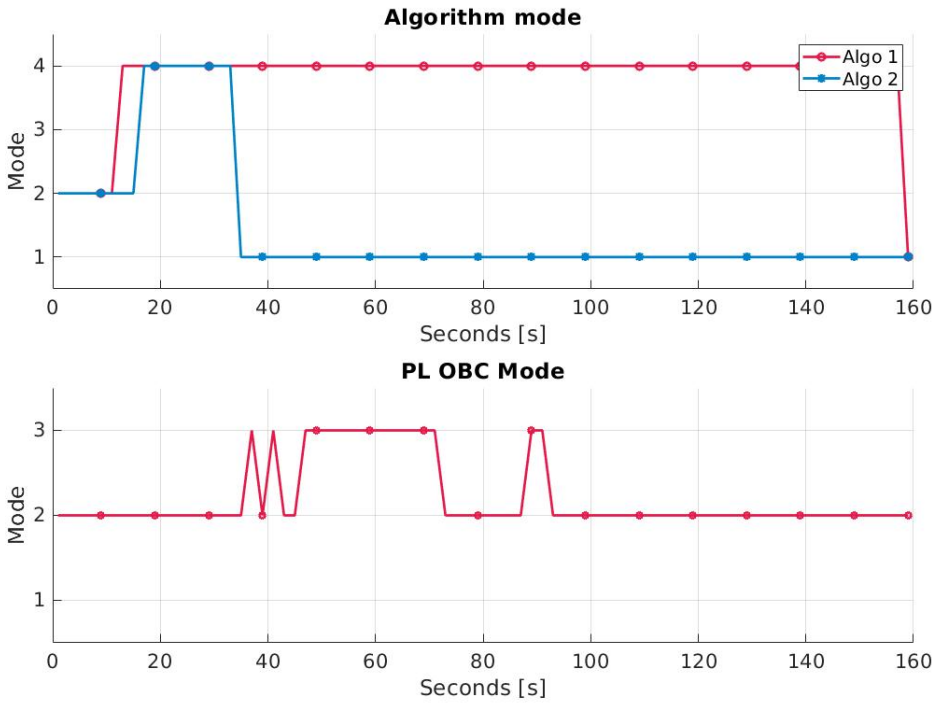


Figure 6.19 – **Electrical energy period** 120 [s] **- Modes over time.**

Figure 6.20 shows an overall good behavior from the HIL setup. For both data types, the number of updates is identical, and the finished results are practically at the same level. Visibly, the timing synchronization problem is nonetheless represented in the results. Surprisingly, the mismatch in data type 1 looks less prominent.

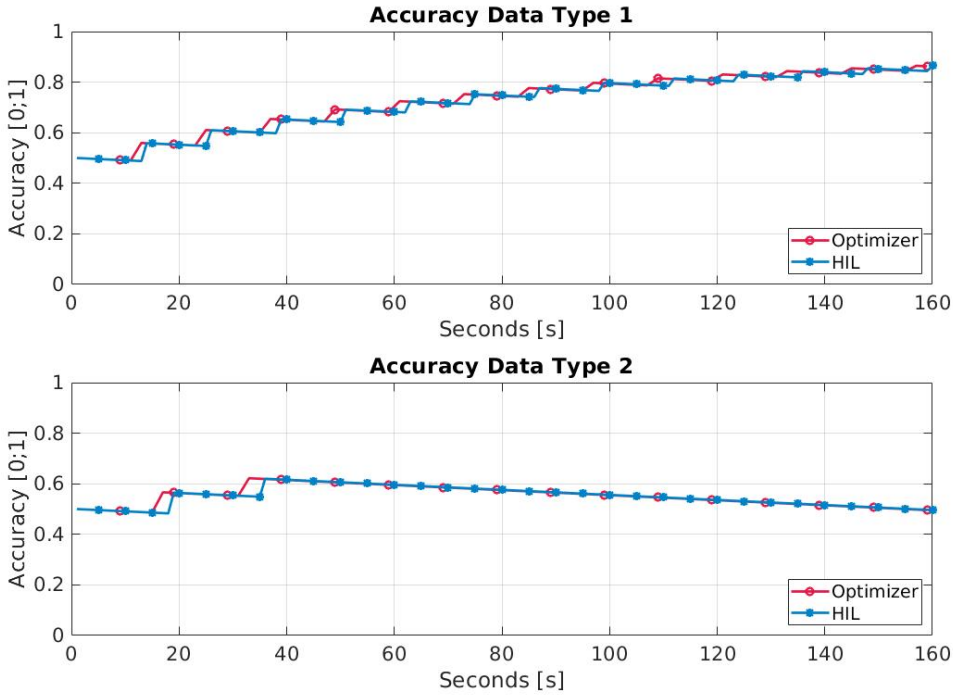


Figure 6.20 – **Electrical energy period** 120 [*s*] **- Accuracy per data type.**

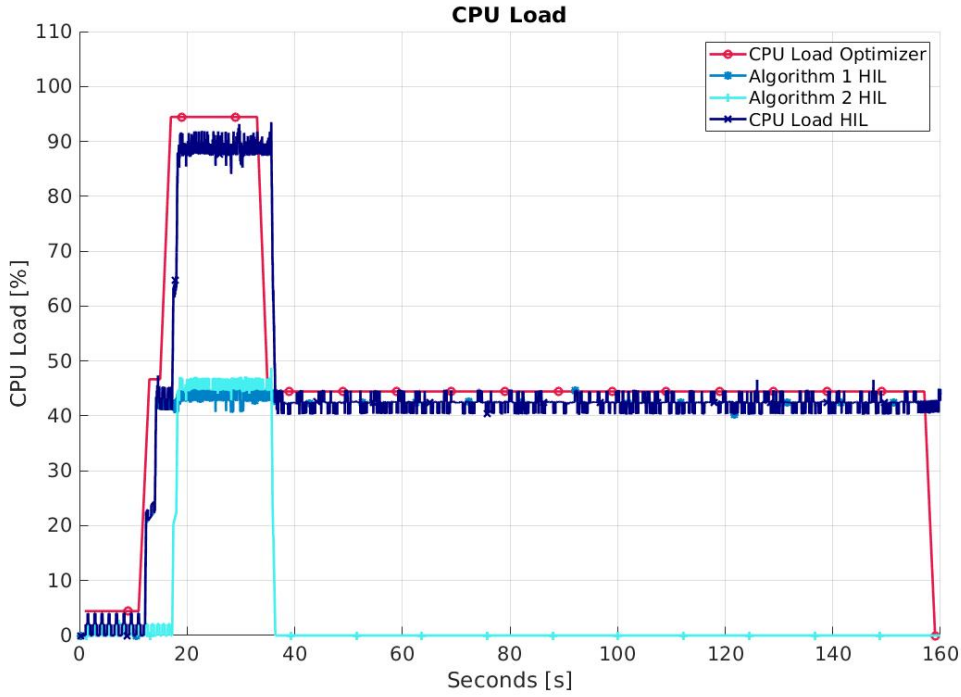


Figure 6.21 – **Electrical energy period** 120 [*s*] **- CPU load.**

Figure 6.21 displays fewer discrepancies between the optimizer and the HIL testbench during the last phase. The peak is experiencing some time delay as well as a lower level, nevertheless, the following phase is more accurate.

PL OBC CPU load

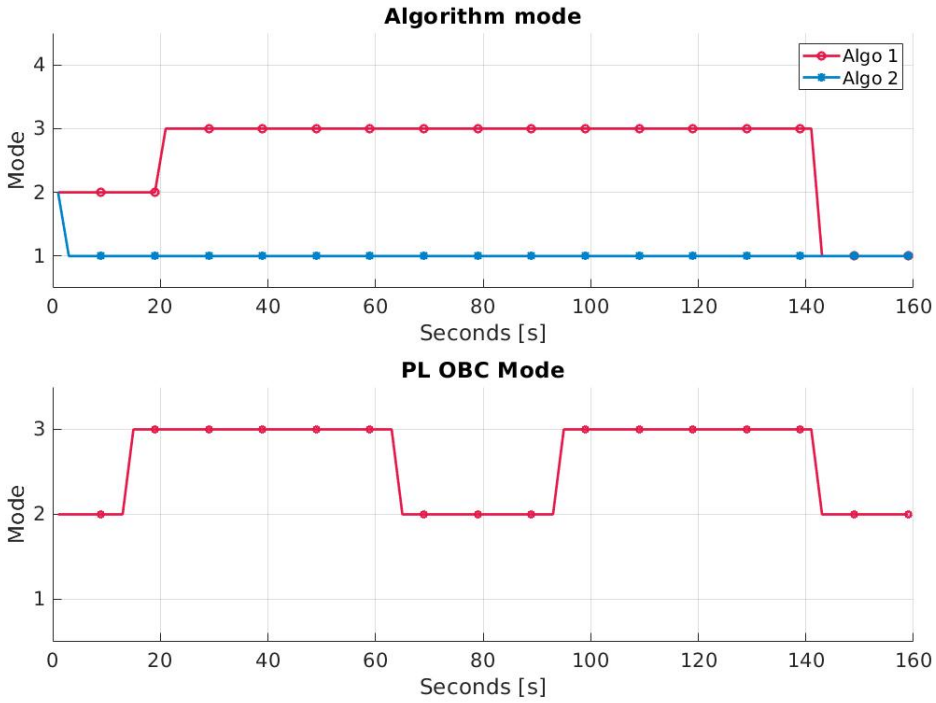


Figure 6.22 – **PL OBC CPU load** 40% **- Modes over time.**

Figures 6.22 & 6.26 contain the mode variation for both cases of the CPU load scenarios. When the overall resources are decreasing to 40% of the nominal case, the PL OBC has three little phases in full state (mode 2) and two longer in mode 3. As consequence, the optimizer picks algorithm 1 to be in a less efficient state (mode 3) and the second to be turned off. This behavior is directly linked to the lack of processing resources to run the algorithms properly. In the other case, the available resources are way higher than usual and the PL OBC varies between mode 2 and 3. This behavior is probably linked to some variation in the sensors' usage. As consequence, algorithm 1 is set in full mode before being turned off. In opposition, algorithm 2 is first set to a less efficient state before the powerful one.

In the first case, Figure 6.23 shows that both algorithms behave the same in the testbench and the optimizer. Their general behavior is identical despite the timing synchronization issue for data type 1. It is problematic to conclude anything for data type 2 since the algorithm is turned off.

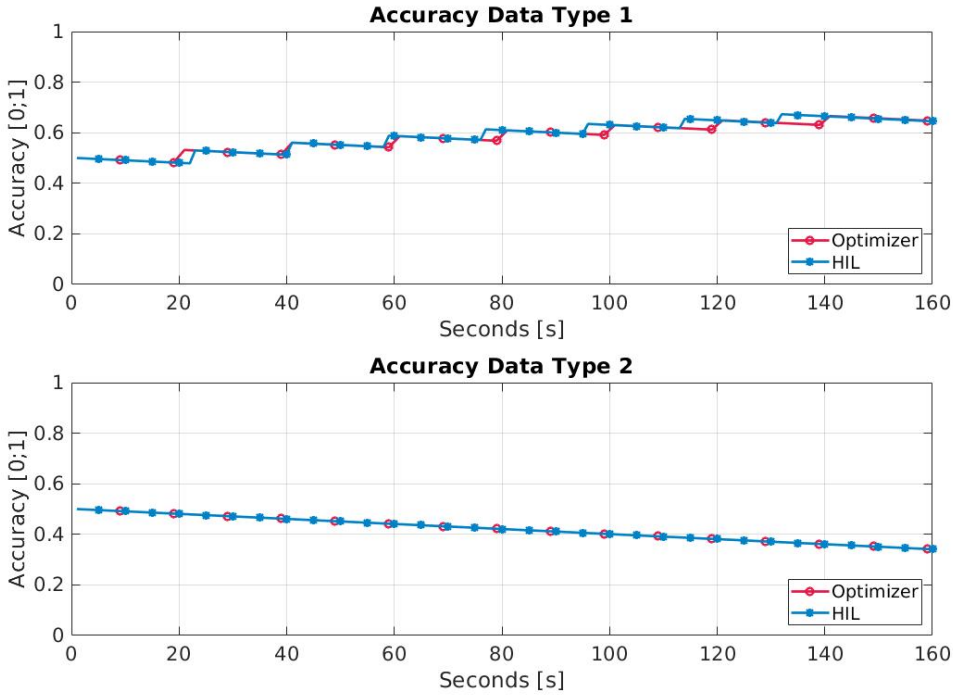


Figure 6.23 – **PL OBC CPU load** 40% **- Accuracy per data type.**

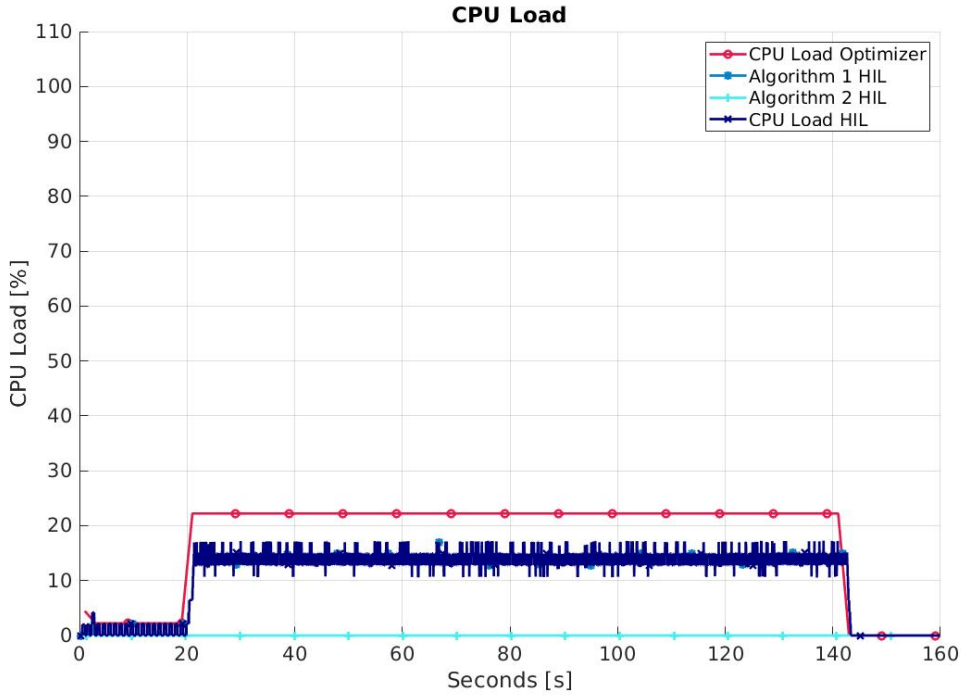


Figure 6.24 – **PL OBC CPU load** 40% **- CPU load.**

The processing resources usage is purely the difference between algorithm 1 from the HIL setup and the optimizer. In Figure 6.24, it can be perceived that the element 2 has no consumption. Nonetheless, the standard lower level of CPU load for the setup is more prominent in this case.

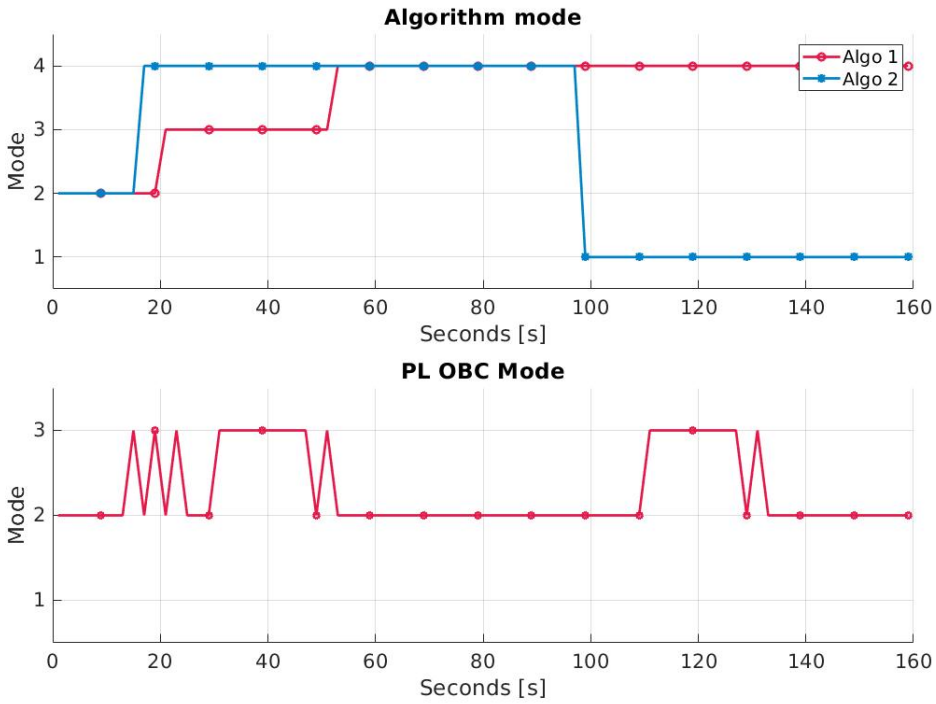


Figure 6.25 – **PL OBC CPU load** 160% **- Modes over time.**

Figure 6.26 is probably the less accurate one with algorithm 2. In this case, the second algorithm is missing a final update due to being shut down prematurely. It results in an ultimate difference in accuracy output. In opposition, algorithm 1 receives the same number of updates for both cases. Nevertheless, it experiences some timing mismatch.

Figure 6.27 has an almost 15% differences in CPU load for the first phase. An odd behavior can be observed with the usage's change of algorithm 2. Even though the mode of the second one is supposed to be fixed at the beginning, it is experiencing a drop of consumption of around 20 seconds in the test. This event is actually happening at the same time as algorithm 1 is altering its mode. It looks like the overall setup has to decrease the resource allocation of both algorithms. This effect seems to disappear during the high consumption phase of the test. The concluding part of the test shows promising results with virtually identical behavior. Predominantly, the trend is similar except for the initial phase.

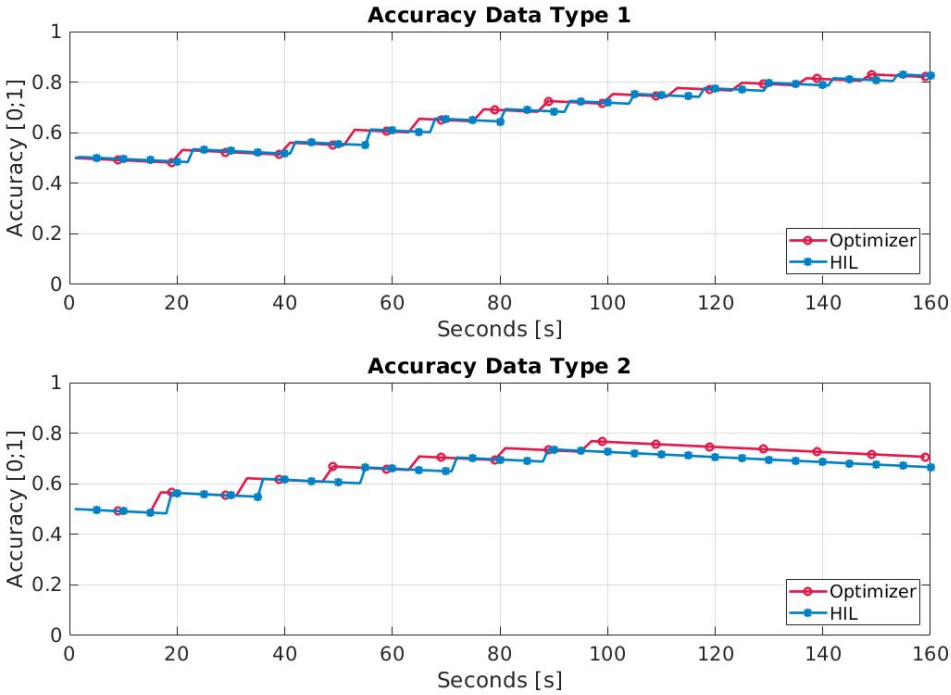


Figure 6.26 – **PL OBC CPU load** 160% **- Accuracy per data type.**

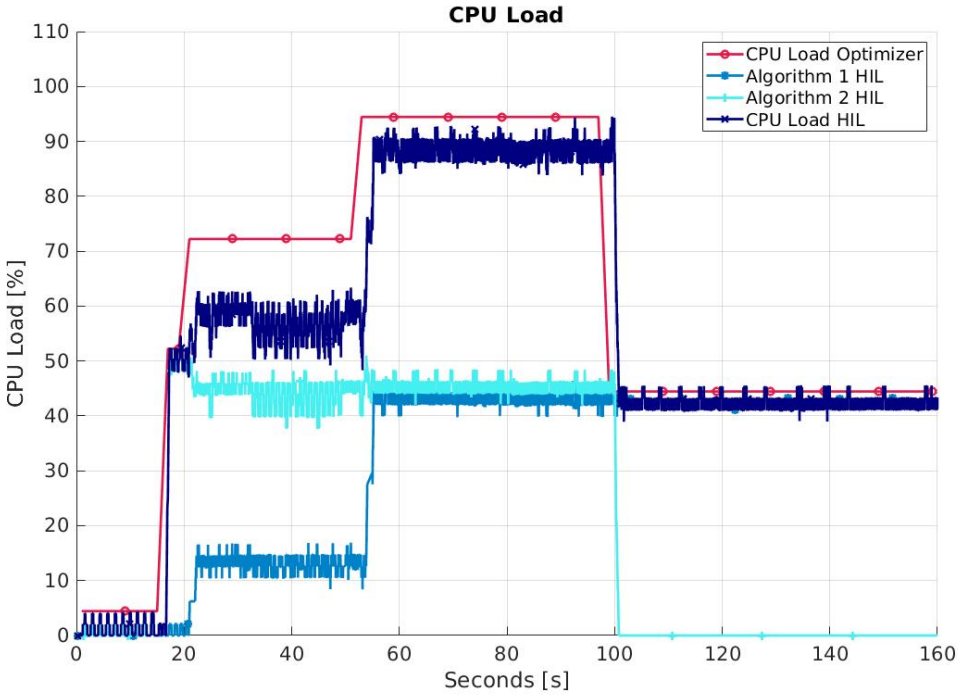


Figure 6.27 – **PL OBC CPU load** 160% **- CPU load.**

3.3 Comments

By analyzing all the various figures presented before, some general comments can be made. Most of them have been mentioned earlier, nevertheless, the goal is to go deeper into their meaning and implications.

The first observation is toward the timing synchronization issue on the accuracy. In many plots like Figures 6.8 or 6.26, a delay in the reception of the algorithm update can be observed. Interestingly, the effect looks to be higher in the case of the second one. This behavior can be linked to the current implementation of the algorithms in the HIL testbench. Indeed, because of the nature of the setup, it is impossible to start both processes (or algorithms) at the same time. For this reason, the new instance of algorithm 2 is created after the first one even if they are supposed to be launched simultaneously. In addition, the launch process implies other tools to be started that will take some time. It is considered a negligible behavior since the monitoring tools will not be needed in an actual architecture. In addition, this effect is not crucial if the algorithms are run for a more extensive period.

A second general comment regarding the overall accuracy is evidently about the algorithm frequency. As seen in almost all the accuracy figures, algorithms in the HIL setup are experiencing a growing delay over time compared to the optimizer. This behavior can directly be linked to the practical implementation of these elements. As explained in the previous section, they are written in ASM with specific timing in their operation. It is easily understandable that matching the period of these mock-up algorithms with their model in the optimizer is not trivial. Slight errors and uncertainties in the implementation can lead to significant discrepancies during the verification. Nevertheless, it is critical to notice that these mismatches disappear as promptly as a similar number of updates have happened. Figure 6.11 represents a typical example where the timing during the update is diverging even though the final results are identical for the HIL setup and the optimizer. Ultimately, it is worth mentioning this issue could be reduced by implementing a more accurate representation of the algorithm in the HIL setup. In addition, it would be more straightforward to analyze an actually implemented algorithm and then modify the model in the optimizer accordingly. This method would ensure a closer match between the HIL setup and the optimizer result.

The analyses of the CPU load (or processing resources) consumed also produce some unusual insight into the verification. The most noticeable one is the usual lower consumption by the HIL setup. In every scenario tested, the algorithm looks to be consuming fewer resources from the processor than expected. A decisive point remains the elevated level of noise and variation that the recording has in the first place. As mentioned earlier, it is fundamental to apply a smooth function to the data to obtain readable outputs. This function can have altered the result in other ways, especially by reducing the average and emphasizing zero reading. Indeed, it has been observed that the recording files have many 0 in their processor usage despite the algorithm being run almost contently. This behavior could explain the typically lower level of CPU usage. Another possibility is a simple mismatch in the implementation of the algorithm

in the ASM code. As for the frequency, the exact processing resources usage is difficult to write in a precise manner. For this reason, the lower level could partially come from these implementation differences. Nonetheless, it is not a critical point for verification. Since part of the mismatch is due to the approximation of the recording tool, a proper margin would be enough to compensate for this. Regarding the mock-up implementation, the problem could be solved by analyzing the behavior of a real algorithm. It would provide insight on the exact CPU load used and then incorporate them into the optimizer model.

On the same topic, it is crucial to mention the time delay between the HIL setup and the optimizer. As stated earlier, this behavior is affecting most of the scenarios studied. Its roots can probably be linked to the delay in the algorithm implementation as explained above. It is possible that the time required to launch the algorithm and start the recording influences the overall plot. Another possibility would be the mismatch between the various clock used in the HIL setup. A few milliseconds could influence and modify the output in a visible way. Even though the cause is less likely, it is a key point to mention. Predominantly, this time delay is not essential to the verification since it stays constant during the whole test. In addition, some tweaks in the launching process inside the HIL setup could resolve this issue.

The ultimate point is the one observed during the last scenario. In some cases, the CPU load of an algorithm is changing despite any other factors. One speculation remains the interference due to additional processes being run in the HIL setup. Even though the processor has 4 cores and uniquely one is used to manage the algorithm, it is possible that other internal processes linked to the OS are happening. In addition, it should be ensured that all the monitoring tools employed are working on another core of the processor. Nevertheless, all these possibilities assumed an actual OS was running on the hardware and that processes need to be monitored. In actual flight hardware, it would likely be replaced by a RTOS strips down from any superfluous program. It would ensure smoother handling of the various algorithms.

4 Outcome & Incoming Challenges

This chapter introduces the first step toward the verification of the optimizer. The current Hardware-In-the-Loop setup has presented promising results regarding both the PL OBC and the algorithm implementation. Thanks to this initial work, the guarantee of feasible and optimized solutions for the avionic platform of ClearSpace-1 is growing.

This verification process represents a crucial milestone toward a broader usage of the tool. By comparing the model embedded in the optimizer with mock-up algorithms in the HIL testbench, it is possible to draw conclusions regarding its validity. The various tests deliver insightful knowledge regarding the exact behavior of the actual algorithms. Emphasis has been placed on the timing and the correct synchronization of the test. No fundamental issues have been discovered during this initial step and future work can continue.

Regarding future development and verification, the sensor parts need to be tackled. Until

now, no realistic models have been implemented to simulate the behavior of real sensors. In addition, the need for hardware parts would be decisive to better understand and compare their behavior. The interaction between the sensors and the memories remains a key point in this verification. In the long term, the addition of hardware sensors in the HIL setup would provide substantial improvement in the verification and validation of the optimizer.

The other essential part requiring verification is the electrical energy management system. In the current HIL setup, the electrical energy is uniquely simulated through the behavior of the processor. It is the first approach to already verify the interaction between the PL OBC and the algorithm in terms of CPU load (or processing resources). The following step would be to govern the electrical energy in the system and look at its effect on the general architecture. It would imply managing the PL OBC but also the sensors in the systems. This additional layer of complexity would move the HIL testbench one step closer to real architecture. Nevertheless, the whole power system might need to be reworked. With this version of the optimizer, the energy constraint is applied only to the payload avionic part and not the whole satellite. It means that it does not include the consumption of other spacecraft elements that can have a major impact on the energy budget of a satellite. In addition, the constraints imposed by the energy variation might not be desirable in a mission planning phase. It would be simpler to upgrade the power system and the size of the solar panel to reduce the energy constraint. In parallel, by distributing the algorithm tasks over time, the optimizer might be able to produce a more efficient solution.

In parallel with this work, it is critical to continue the development and improvement of the optimizer. In one direction, the goal is to test the optimizer with a broader variety of scenarios and architecture. By looking at the past and upcoming missions in the formation flying and docking domain, a considerable number of possibilities can be found. The idea would be to compare the result of the optimizer with fitting constraints relative to the actual architecture used in the requirements. It would permit to thoroughly understand the various requirement of such missions as well as to test the robustness of the optimizer. The goal of the next chapter is to implement a realistic case of an ADR mission.

In summary, the results presented in this chapter are promising for the future of the optimizer. The verification of behavior regarding the algorithm and the PL OBC shows the capabilities of the tools for subsequent applications.

7 Practical Application

1 Introduction

The previous chapters have shown the sensitivity of the optimizer to some parameters and the verification of the mathematical model. This chapter is dedicated to the application of the optimizer to a relatively realistic scenario. The goal is to create a model for the PL OBC and its sensor with values close to real hardware components. To achieve this, an analysis of various instruments is done and then an example of payload avionic is modeled. The next step is to choose a flight phase of an ADR mission and extract the general constraint. By inputting them into the tool, the optimizer's results will provide meaningful information relative to a potential mission planning scenario for this flight phase. In summary, this chapter introduces an example of the optimizer usage in the case of an Active Debris Removal mission.

2 Sensor Analysis and Energy Consumption

This subsection is dedicated to the analysis of the rendezvous sensors and their electrical energy consumption. The goal is to better understand the actual requirements of such elements and apply them to the optimizer. It is a synthesis of an earlier study of the author[169] that describes each category of sensors and their outputs.

Since the CS-1 mission is still in an early development phase, different trade-offs exist for the types of sensors that need to be used to accomplish the rendezvous and capture. These trade-offs almost exclusively consider the type of information that each sensor can provide. At this point, multiple manufacturers and even universities have been considered components suppliers. To obtain some metrics, an assessment of sensors characteristics in terms of energy and data rate has been made. This analysis allows defining quantitatively more precise requirements for the PL OBC and, in turn, will be useful for the scenario presented below.

The first category of sensors is optical cameras (visible light). For the mission, these sensors are valuable during the target inspection and line-of-sight navigation. To assess the data rate,

the first metrics to retrieve is the resolution of the sensor as well as the number of frames per second. The following step is to find the pixel encoding of the sensor or the number of bits per pixel. For the encoding, the standard of visible cameras is ordinarily 24 $[bit]$ per RGB pixel. Concerning the two other parameters, they are specific for each sensor and can be obtained through datasheets. The ultimate key metric is electrical energy consumption. It must be noted that the interface/connector remains a crucial point, even though this element can often be adapted and changed. Finally, the other camera parameters like pixel size, field of view, mass, dimension, etc. are not considered for this analysis. The manufacturers that have been looked at are primarily Photon Focus[170] and TSD Space[171]. Results show that on average the visible imaging sensor roughly has a 3.8 $[Gbit/s]$ data rate and an energy consumption of 5.4 $[W \cdot h]$. It is critical to remember that Photon Focus does not make space-grade components by default. In this manner, their performances may decrease by several factors if components qualification is required.

The second category is thermal infrared cameras. They serve the same purpose as the visible ones except that they are less sensitive to illumination conditions. Because of their nature, their characteristics are similar to the visible cameras. The fundamental differences reside in the thermal sensitivity which remains an essential parameter for these types of sensors. Nevertheless, the resolution, frame per second, electrical energy, and interface are the needed parameters. For the case of thermal-infrared sensors, the encoding is typically 14 $[bit]$ per pixel. By gathering data from companies like OPAL[172] and Ulis[173], it gives an average of 23.9 $[MPixel/s]$ or a data rate of 330 $[Mbit/s]$. The energy consumption averages around 1.8 $[W \cdot h]$.

For the imaging sensor, the LIDAR technologies provide equally pertinent information. They can extract ranging and be used for motion reconstruction. The metric for LIDAR technologies is a bit different than for standard imaging sensors. The manufacturer typically considers data rate as the number of points per second. Indeed, because of the operation of such devices, it makes less sense to consider the number of pixels per image. As previously said, the interesting parameters are the ones linked directly to data rate and electrical energy. Regarding the quantity of information, data points are typically composed of an intensity ($12-14$ $[bit]$), a position in space (Coordinate x, y, and z with 4 $[Byte]$ each), and a timestamp (8 $[Byte]$). This means that each measurement is encoded with around $172-174$ $[bit]$, which is significantly higher than for other sensors. For the data rate, LIDAR technology usually has a rate of a hundred of thousand points per second. It creates an output of around 55.7 $[Mbit/s]$ with a usual energy consumption of 100 $[W \cdot h]$. These data come from OPAL[172] and Jenoptic[174] sensors. It is interesting to notice that the CSEM[175] has developed a LIDAR with lower performances. It has an average data rate of 0.66 $[Mbit/s]$ and a peak energy consumption of 5.2 $[W \cdot h]$. Its mass is divided by a factor of 10, and its range and accuracy are reduced.

Concerning the Star Tracker, it is assumed that they can also be used as regular cameras. It is a good way to repurpose them when they are not needed by the main spacecraft. It also allows for saving the mass and complexity of having additional sensors. Nevertheless, the difficulty

Type	Average pixel rate [MPixel/s]	Bits per pixel	Average data rate [Mbit/s]	Average Energy consumption $[W \cdot h]$
VIS Camera	160	24	3810	5.5
IR Camera	24	14	334	1.8
LIDAR	0.3	~ 174	42	75
Star Tracker	~ 200	16	~ 3′000	5.6

Table 7.1 – **Summary of sensor analysis.**

was to assess the non-processed data output of the sensor. The primary function of these elements is not the point of interest here. These instruments output quaternions computed by their own processor and the manufacturer does not provide information on the imaging sensor. It means the analysis of the exact raw data provided is not possible. In addition, most of them employ methods like Region Of Interest (ROI) to reduce the amount of information that needs to be processed. For the metric itself, it can be estimated that the data rate is the same as a high-performance imaging sensor in the worst case. Ultimately, Star Tracker could potentially provide the same information as the other imaging sensors: target inspection and line-of-sight navigation if they are used as cameras.

The table 7.1 represents a first approximation of the electrical energy usage and data rate of the various categories of sensors. These numbers represent the average between manufacturers and do not consider any encapsulation or formatting in the data. An internal *.xls* file has been created with the synthesis of the information presented. This file provides a refined approximation of each category of sensors and also contains information on the various models. In some cases, it additionally includes the maximum and minimum rate and energy consumption. This information is crucial to establishing a realistic sensor model for the scenario presented below.

3 PL OBC Architecture

For this relatively realistic PL OBC architecture, the goal has been to keep this usage example at a basic level. In this case, it uses three instruments with cold redundancies and has two main algorithms running in its PL OBC. Figure 7.1 shows the layout of the payload avionic (the redundancy is not shown here).

Regarding the sensors, there are two cameras with one operating in the visible range and the other one in the thermal range. In addition, the payload avionic includes a LIDAR. The first camera is based on the model MV1 from Photon Focus[170]. The camera needs 12 $[V]$ to operate and uses 24 $[bits]$ per pixel. It has been decided that the camera can have four different modes with the first one being turn-off. The second one is the LE mode with an electrical energy consumption of 1 $[W \cdot h]$ and a resolution of $1024x1024$ pixels with one image every 8.5 $[s]$. It means an average data rate of 3 $[Mbit/s]$. The third mode refers to a fast operating configuration consuming 5.4 $[W \cdot h]$. It has again a resolution of $1024x1024$ pixels

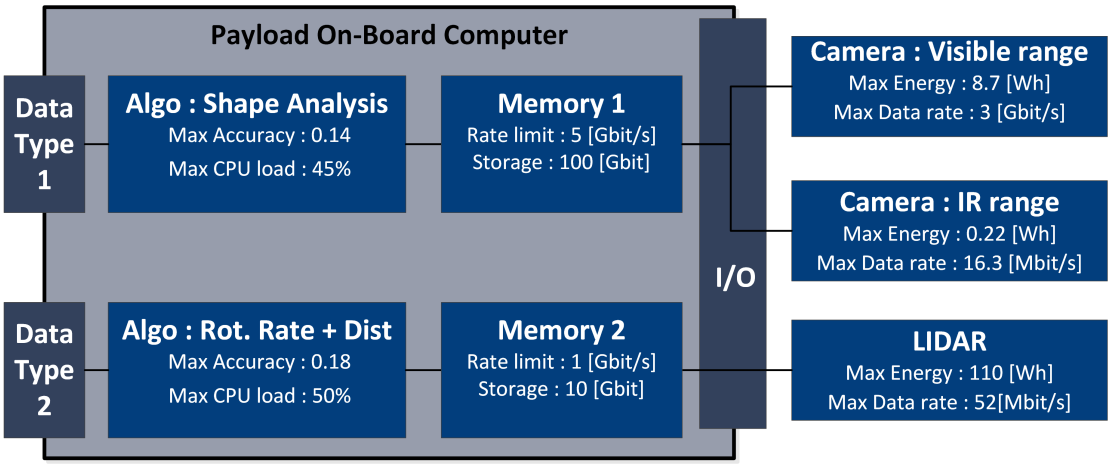


Figure 7.1 – **Example of realistic payload avionic.**

and a frame rate of 34 which means an average data rate of 857 $[Mbit/s]$. Finally, there is the precise mode that uses 8.7 $[W \cdot h]$. It has a data rate of 3020 $[Mbit/s]$ coming from a resolution of $4096x3072$ pixels and a frame rate of 10 images per second.

The second camera is a derivative of the Ulis[173] PICO160Gen2 model. It has a fixed resolution of $160x120$ pixel with a 14 $[bit]$ per pixel encoding. It is assumed to have three modes with the usual turn-off one. The LE consumes 0.1 $[W \cdot h]$ and has a frame rate of 9 which creates an approximate data rate of 2.5 $[Mbit/s]$. The operation mode has an electrical energy consumption of 0.22 $[W \cdot h]$, a frame rate of 60, and thus a data rate of 16.3 $[Mbit/s]$.

The LIDAR is inspired by the OPAL-P500[172] product and is assumed to have four modes with the first one being the turn-off. It encodes every measurement point with 174 $[bit]$. The LE mode needs 10 $[W \cdot h]$ and has a data rate of 2 $[Mbit/s]$ since it produces $10k$ points per second. The minimal operating mode consumes 40 $[W \cdot h]$ and outputs $90k$ points per second (16 $[Mbit/s]$). Finally, the operation mode requires 110 $[W \cdot h]$ and produces $300k$ points per second (52 $[Mbit/s]$).

As shown in Figure 7.1, the visible and IR cameras are linked to the first memory which has a storage capacity of 100 $[Gbit]$. The third instrument is linked to the 10 $[Gbit]$ memory.

The PL OBC is derived from the GR712RC development board developed by Cobham Gaisler AB[49] used in the two first chapters of the thesis. It is assumed to have three modes. The first one is the LE mode with an electrical energy consumption of 0.2 $[W \cdot h]$ and 10 $[MIPS]$ of CPU load available. The minimal operating mode consumes 0.5 $[W \cdot h]$ and provides 110 $[MIPS]$ of processing resources. Finally, the operation mode needs 1.3 $[W \cdot h]$ and has 200 $[MIPS]$ available.

Inside the PL OBC, two distinct algorithms are running. The first one is dedicated to the shape analysis of the target and uses the information from the first memory. Its purpose is to

reconstruct a potential 3D model of the targeted debris with images from visible and infrared cameras. The goal would be to better understand the current shape of the target and identify potential hazards before capturing it. To represent this, the accuracy parameter defines that 1 is a perfect model of the target. This algorithm includes four different modes and the first one is simply the turn-off. The LE mode needs 4.5 [$MIPS$] (2.25% of max CPU load) and an average of 5 [$Mbit/s$] of data. It has a period of 100 [s] and increases the accuracy of the 3D model by 0.01 at every update. The minimal operating mode requires 45 [$MIPS$] (22.5% of max CPU load) and 600 [$Mbit/s$] of data. The period is set to 50 [s] and the accuracy output to 0.09. Finally, the operation mode needs 90 [$MIPS$] (45% of max CPU load) and an average of 2000 [$Mbit/s$]. It increases the accuracy by 0.14 and is updated every 40 [s].

The second algorithm has the main purpose of estimating the rotational rate of the target as well as the distance from the ADR satellites. For this, it needs the information from the second memory and thus the LIDAR. It is important information for the chaser to enable proper navigation and avoid collisions. The rotational rate is also decisive for the mission planning to coordinate the capture later on. The accuracy parameter describes the confidence of the algorithm about the attitude of the debris at any moment with 1 being a perfect knowledge. As for the previous algorithm, it contains four distinct modes and the first one is simply the turn-off. The LE mode needs 4 [$MIPS$] (25% of max CPU load) and an average of 2 [$Mbit/s$] of data. It has a period of 100 [s] and increases the accuracy of the distance and rotational rate by 0.01. The minimal operating mode requires 55 [$MIPS$] (27.5% of max CPU load) and 30 [$Mbit/s$] of data. The period is set to 50 [s] and the accuracy output to 0.1. Finally, the operation mode needs 100 [$MIPS$] (50% of max CPU load) and an average of 80 [$Mbit/s$]. It increases the accuracy by 0.18 and is updated every 40 [s].

The last point regarding the payload avionic architecture is about the two data types. As mentioned, the first one is regarding the shape of the target with a 3D model. For this one, an arbitrary α parameter is set to 0.001. It is a relatively small number since the information about the shape is very unlikely to change and degrade over time. The second data type is the rotational rate and distance to the target. A value of 0.003 is given to this α parameter. It is assumed that the information can degrade more rapidly over time if no update is provided by the algorithm. In addition, this algorithm could be more subject to error propagation because of the orbital mechanics.

4 Scenario

For the scenario, it has been decided to use a specific phase of a typical ADR mission which would be the inspection of the target or fly-around. Figure 7.2 shows an example of the operation. For the inspection, it is assumed that the chaser will slowly orbit around the target in about 10 [min] while being exposed to the sun. This rotation around the target allows for studying all the faces of the debris with multiple instruments. For this phase, it is assumed that the chaser has its batteries charged. In addition, the satellite already has some initial

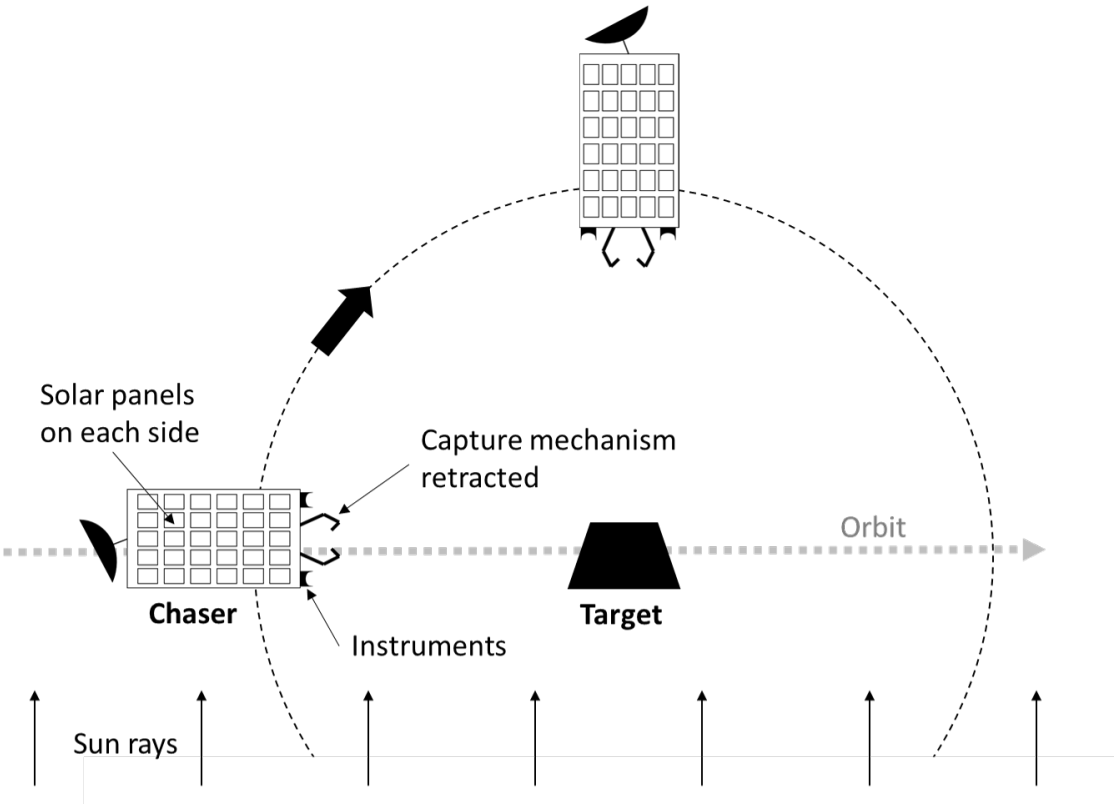


Figure 7.2 – **Example of mission phase scenario.**

value for both data types with a starting point of 0.2.

This orbit maneuver is performed passively without any thruster. The rotation around the target implies some change in the solar panels' exposition to the sun. (The satellite has solar panels on its four longitudinal faces.) As shown in Figure 7.2, the alignment with the sun will be perpendicular at the beginning, after 5 $[min]$ and at the end which means the maximum of electrical energy production. For this reason, the electrical energy production dedicated to the payload has an offset of 120 $[W \cdot h]$ and an amplitude of 30 $[W \cdot h]$. It is assumed that the consumption of the main platform is already subtracted from this electrical energy amount and with a proper margin.

Even though the goal is to inspect the target, a slight emphasis is placed on the second data type (rotational rate and distance). It is important for the avionic to know at every moment these two parameters. For this reason, the weights are respectively 0.9 and 1.0.

5 Results

The section is dedicated to the result of the scenario presented above. The goal is to present and describe the various figures. The evaluation of the results is done in the next subsection.

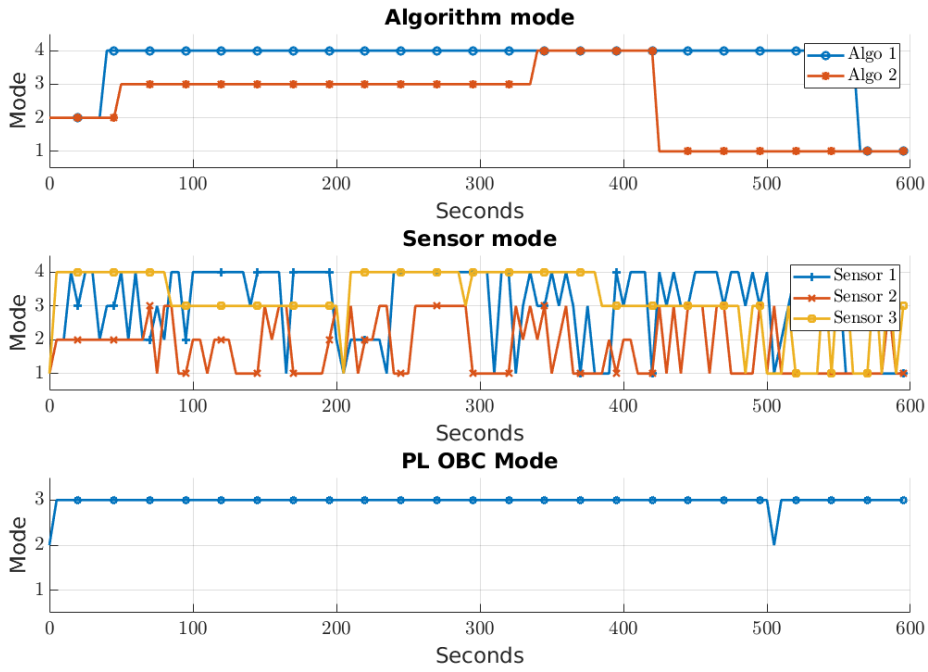


Figure 7.3 – **Fly-around scenario - modes.**

Figure 7.3 shows the mode's variation of all elements inside the payload avionic. Regarding the algorithms used, their respective behaviors have not a lot of variation. After some times in LE mode for both of them, the 3D modeling one is set to its operation mode for the full duration. In comparison, the rotational rate and distance algorithm is first set to its minimal mode for almost 6 $[min]$. Then, it is set to its operation mode but during slightly more than 1 $[min]$. Finally, the algorithm is simply turned off. For the sensors, the results have the same high-frequency fluctuation as in the previous chapters of the thesis. Nevertheless, the trends show that the first sensor (visible range camera) is more often turned on than the infrared camera. The LIDAR seems to be working for the first three quarters of the scenario. Finally, the PL OBC stays in its operation mode for the full duration of the scenario with one unexpected drop to its minimal mode.

Regarding the outputs' accuracy, Figure 7.4a shows their evolution over time. As a reminder, the data type 1 refers to the 3D model of the target. In this case, the accuracy of the model keeps increasing during the full scenario. As explained in one of the previous chapters, the effect of the update decreases as the accuracy itself increases. Indeed, taking a new picture after 8 min of analyzing the target will not have the same impact on the model as the ones taken in the first minutes. The accuracy slope for the rotational rate and distance (Data Type 2) is interesting. It is first updated at regular intervals until about 7 $[min]$ when the curve simply decreases due to algorithm 2 being turned off. It is also possible to notice the differences in the α parameter between the two data types with their decreasing slope.

Figure 7.4b displays both the memory usage and the data rate at the memory. The high fluctuation of the memory 1 is directly linked to the high data rate of the camera in the visible range. As explained, this camera is able to produce up to 3 [$Gbit/s$]. For this memory, the infrared camera has almost no impact with its maximum of 16.3 [$Mbit/s$]. The second memory also has some variation, but it is less visible in the figure. Regarding the data rate, the maximum is never attained and thus the payload avionic is not constrained by this factor.

The electrical energy consumption and production are shown in Figure 7.5a. The purple curve indicates what is not being consumed by the payload avionic. As a reminder, the energy available shown here does not reflect the actual energy budget of the entire satellite. The blue curve indicates what is provided to the payload avionic by the power system of the satellite. It means effects such as battery charging and consumption of others elements are not hidden. Overall, it can be observed that the payload avionic reaches on a few occasions a 0 [$W \cdot h$] balance. It implies that the system is constrained by this element.

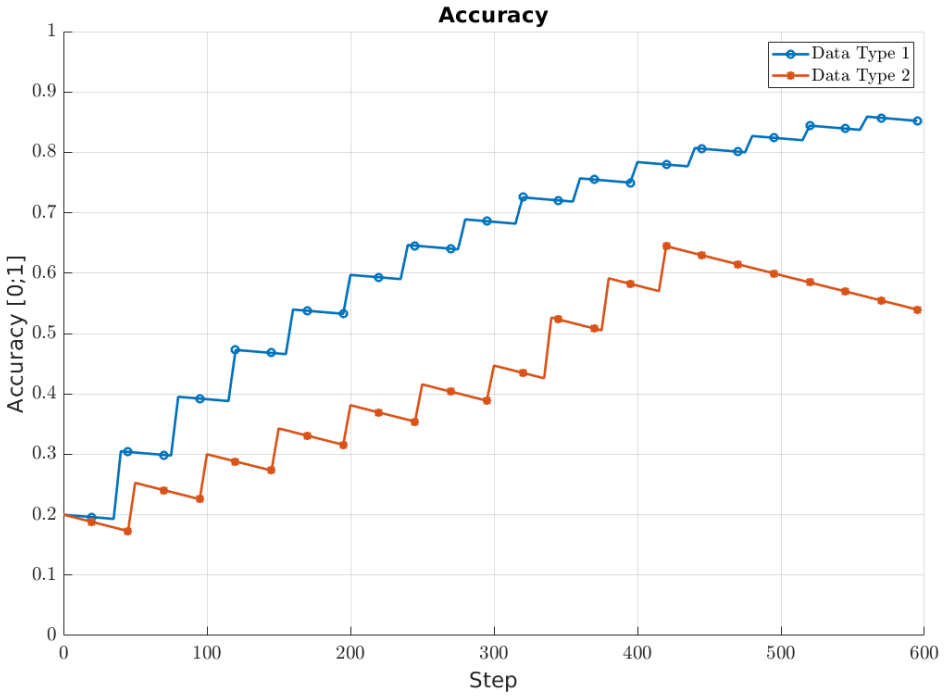
Finally, Figure 7.5b shows the CPU load in the PL OBC. As it can be seen, the available resources are enough to cover all the phases of the scenario. With the consumption curve, it is also possible to distinguish the change of mode of the two algorithms. The drop of MIPS available in the PL OBC is directly linked to its change of mode.

6 Solution Analysis

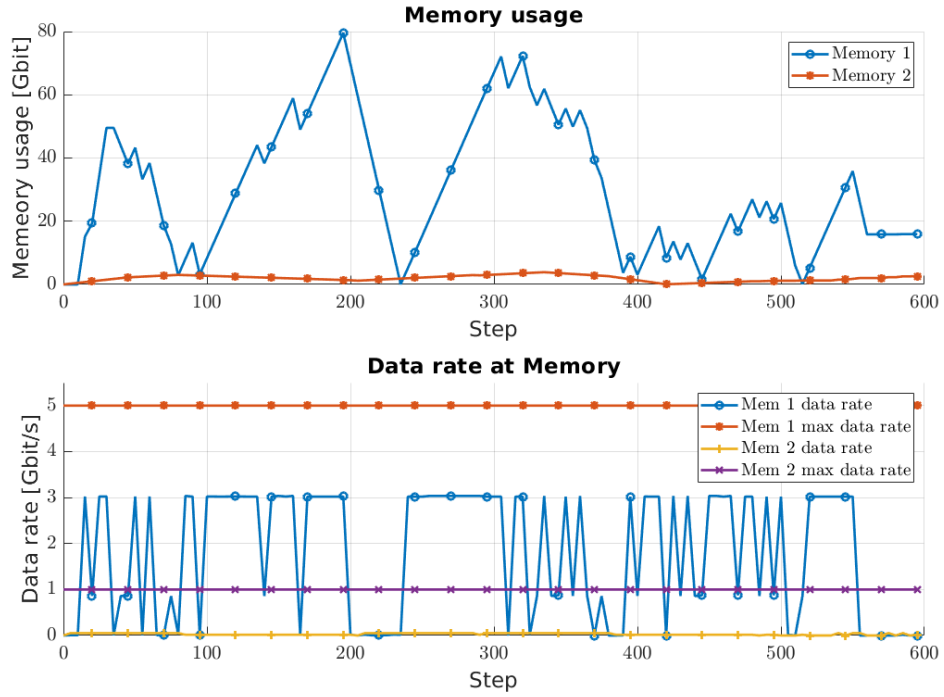
The various figures shown in the previous section can be analyzed in greater detail and the overall behavior of the payload avionic can be understood. As a first point, it is important to remember that this scenario and payload avionic architectures are fictive despite being inspired by real hardware elements. It is a well-suited example to understand the strength and weaknesses of the optimizer.

On the general level, the optimization has partially succeeded in its task. Indeed, the goal of the scenario is to inspect the target with a fly-around while maintaining fundamental information about it such as distance and rotational rate. Figure 7.4a shows that the second part is not achieved since the distance and rotational rate are not updated for the three last minutes. In opposition, the 3D modeling of the target seems to be achieved with an increase from a 0.2 to 0.85 accuracy level. For a tool user, it is important to first find the root of the issue before suggesting some modification to the architecture and scenario.

The lack of update in the final part of the scenario is obviously linked to the second algorithm being turn-off (Figure 7.3). The optimizer has taken this decision probably for a specific reason. Some clues can be found in the memory usage plot (Figure 7.4b) by zooming on the second element. This graphic shows that the second memory is actually empty at around 7 [min]. This implies that the second algorithm can not be run because of the lack of data. By looking at the LIDAR (sensor 3), it can be observed that it is set in its minimal operation mode and then being turn-off on multiple occasions starting at around 6 [min] (Figure 7.3). It means a

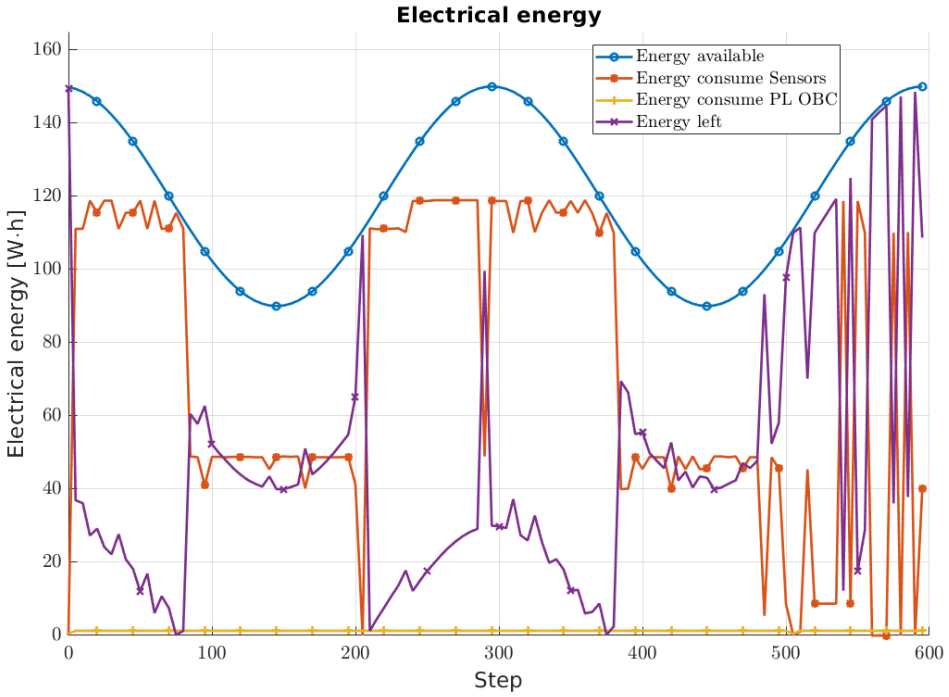


(a) Accuracy of output data.

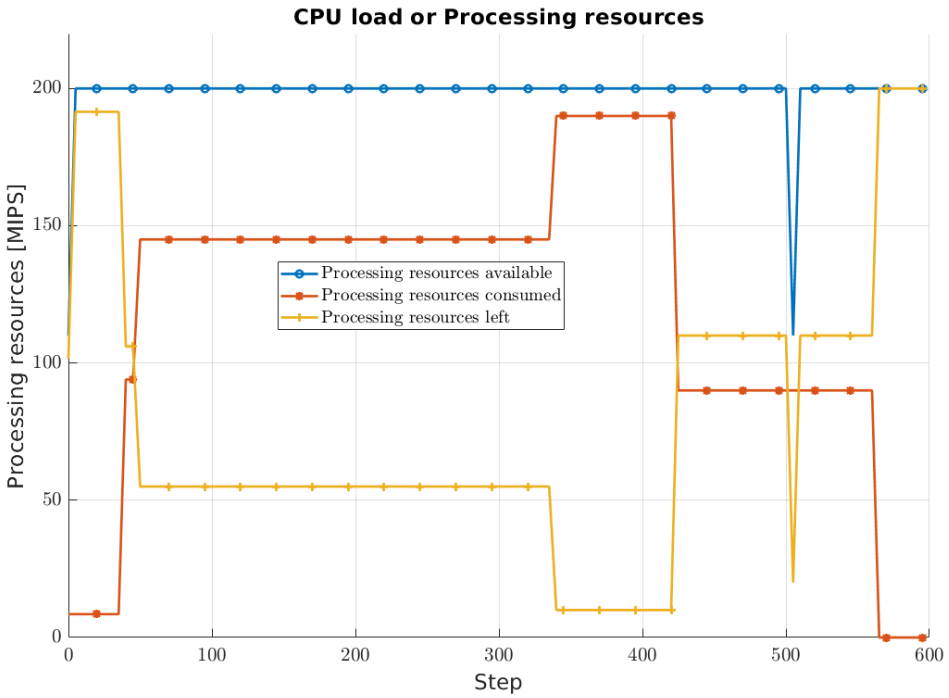


(b) Memory usage and data rate at memory.

Figure 7.4 – **Fly-around scenario 1 - Part 1.**



(a) Electrical energy.



(b) Processing resources.

Figure 7.5 – **Fly-around scenario 1 - Part 2.**

lower amount of data is produced and thus the memory is slowly emptying. At this point, the issue of the LIDAR can be traced back to the actual electrical energy budget in the payload avionic. The 7 [min] marks is close to the minimal electrical energy production and thus the optimizer has to make some choices. Indeed, the LIDAR has by far the highest electrical energy consumption of the whole system, and it is probably why it is the one affected. In addition, since it is the unique instrument connected to the second memory, no other element can take over.

As a tool user, it is now possible to create a few hypotheses on how to implement modifications to tackle this issue. The first and most straightforward one would be to increase the electrical energy budget allocated to the payload. This simple effect could solve most of the issues and allows to have both algorithms running all the time. Nevertheless, this solution might not possible since it would imply modifying the spacecraft or increasing the electrical energy budget allocated to the payload. The second solution would be to change the LIDAR instrument with something that consumes less electrical energy. It could be either by choosing another manufacturer or changing the type of sensor. In both cases, it implies some hardware changes that might not be possible. The last solution would be to modify the weight associated with the two data types in the optimizer. By increasing the importance of the distance and rotational range algorithm, the optimizer might be able to find a different solution. Of course, it would probably imply a decrease in accuracy regarding the target modeling. The three solutions exposed are the most straightforward ones and more could eventually be found to tackle this issue.

7 Discussion

As shown in the previous section, the optimizer does not provide a perfect solution but suggestions for a scenario following the constraints imposed. It is then the role of the user to find the appropriate solution to its problem and adapt the payload architecture if needed. The next step would typically be to run again the simulator with a different configuration and check if the requirements are met. It could be as well to move to more in-depth hardware and software testing if the result from the optimizer is satisfactory.

Despite interesting results, the tool still suffers from particular behaviors. A typical example is the mode's change of the PL OBC which happens somewhere between 8 [min] and 8.5 [min]. This sudden modification is difficult to explain, especially with the low energy consumption of this item. As it has been shown, this electronic board consumes almost two orders of magnitude less energy than the LIDAR. It means its mode has almost no effect on the general electrical energy budget. Moreover, the CPU load (Figure 7.5b) shows that the resources consumed are not affected by this modification which is technically impossible. The most probable reason is that the optimizer is actually suffering from a small design error. More investigation would need to be carried out. Another issue that has already been talked about in one of the previous chapters is the fast variation of the sensors' mode. Even though the time

step in this optimization is way longer, it still implies variation of mode as short as in 5 [s]. As explained, these variations are coming from the lack of constraints regarding the sensors' usage. It is something that needs to be updated to obtain proper results.

Regarding the design of the scenario, some comments could be made about the lack of margin in the CPU usage. As mentioned in one of the previous chapters, algorithms consuming almost all the processing resources of the PL OBC are simply not acceptable at the mission design level. Proper margins should be implemented and thus the consumption of the algorithm would need to be decreased. Nevertheless, it is a straightforward implementation in the optimizer since it implies uniquely modifying a few numbers to match a proper CPU load budget.

As a general comment, the optimizer has a large number of constraints implemented. This fact does not mean the user has to employ all of them for its design. For example, the scenario shown is not constrained by many factors and it does not represent an issue. It is the decision of the user to set the requirements properly for its application and write the correct model.

One major drawback of the optimizer is its desire to make compromises between various algorithms and sensors during a short period of time. Indeed, by design, it will try to run all elements it has to maintain a decent level of accuracy on all the data types. As seen with the example, the design and thus the results were highly driven by constraints from the electrical energy for this reason. It is something that should not happen in a real mission plan and the requirements should mainly come from the operation needs. Evidently, the planning of such missions will always face some constraints from the energy budget since it is probably not possible to run all the hardware elements of the spacecraft at the same time. It is the role of the tool user (or mission planner) to already restrain the number of elements that need to be used per phase.

Nevertheless, it might be needed to have a few instruments running at the same time in some situations. In the scenario shown, the goal was to reconstruct the 3D model of the target while keeping the spacecraft informed about the distance and rotational rate of the target. Both data types were needed at the same time, even though some compromises could be done regarding the load of the algorithms and the energy consumption of the sensors.

In some cases, another solution would be to split the task inside the PL OBC among multiple orbits. An example could be to have the 3D modeling during one orbit and then the tumbling analysis done during the following one. This task allocation would reduce the load on the payload computer as well as the constraint on the electrical energy budget. As presented in the literature review, it is typically what has been done with TINA System[143]. This specific tool allows the distribution of Earth observation satellites' tasks among multiple orbits. Regarding the optimizer presented, the avionic payload model could be altered to allow a similar procedure as TINA while focusing on a specific phase of the mission. The optimization could be limited to a small duration of time when it is decisive for multiple algorithms to be run at the same time. In any case, it is important to remember that the optimizer provides a potential

solution for the mission planning following the constraints imposed. It is the task of the user to model the correct payload avionic to satisfy all the requirements imposed by the mission.

8 Outcome & Incoming Challenges

This chapter has shown the potential of the tool with a realistic example. A typical process design has been demonstrated following these steps. First, a payload avionic architecture has been modeled following documentation from actual hardware elements. Second, a scenario inspired by the typical fly-around mission phase has been created. Third, the result of the optimizer had been analyzed and it has been estimated that it did not meet the requirement of the flight phase. Fourth, the root of the problem has been found and solutions have been suggested to modify the payload avionic architecture. This process loop can be iterated multiple times to converge to an appropriate design.

The optimizer used in this chapter still requires improvement and modification. Some elements are still subject to unexplained behavior and might alter the feasibility of the solution outputted. In addition, the level of abstraction in the tool remains relatively high. Even though more precise models have been used in this chapter, the dynamics and specificity of the tool do not allow a finer representation of the payload avionic interaction. The addition of features and the modification of some parts of the tool are required. One of the key elements is the management of the electrical energy budget and the constraints it applies to the solution. Finally, the type of optimization might need to be modified to output more realistic solutions. Tasks allocation is one of the most straightforward solutions to tackle some issues of the tool.

8 Conclusions & Future Work

The development and implementation of complex avionics for Active Debris Removal missions are still in their infancy. To this day, it can be considered that autonomous rendezvous in space have been mastered typically looking at the newcomer in the industry. Nevertheless, the uncooperative part remains a relatively untouched domain with no practical demonstration. Missions have been launched and technologies have been evaluated but no real debris removal has happened yet in LEO. One of the decisive aspects of success in uncooperative rendezvous is the flexibility and reliability of the avionic system. The challenge of this Ph.D. thesis is to deliver tools and analyses to tackle the space debris issue. The driving factor is to accelerate the development of ADR missions with an emphasis on the ClearSpace-1 case.

One of the first aspects presented in this work is the relation between the main platform processor board or avionic with the payload computer. The key task of the Payload On-Board Computer (PL OBC) is to gather data from various sensors and use its internal algorithms to extract meaningful information about the target. These computed data are required by the Platform On-Board Computer (PF OBC) to plan its trajectory and react accordingly to the state of the debris. The link between these two avionic boards is considered crucial.

1. To address this aspect, I initially designed a robust data exchange protocol. It has to allow efficient interaction between the Platform On-Board Computer and the Payload On-Board Computer. Moreover, it should be resilient against various connection issues and data shortages. In the first part of the work, a high-level protocol using SpaceWire/RMAP and Ethernet is suggested. The primary advantages are the inherent robustness of the communication bus used and the high broadband available. Implementing this data bus also implies some additional constraints with the packet encapsulation required for the correct routing of messages. Furthermore, a study is conducted on the timing needed by the main platform avionic to obtain information from the payload computer. A fundamental design is established regarding the structure of the packets and the data required to be sent on the link. A defined architecture is chosen with the flexibility to deliver multiple types of information. In addition, a verification procedure is created

regarding the receiving packet. It is a decisive point to ensure the correctness of the information received. Ultimately, a prototype software is developed for the PL OBC to simulate the actual behavior of the final version. This initial study lays the path for subsequent developments of the data exchange protocol. It creates a robust base for the assessment and testing of this essential board interconnection.

2. For the second work, I continued on a similar path with analyses of other communication data buses. The driving point is to ensure a reliable backup or fall-back solution in case the primary link fails. It additionally offers an easier alternative for data exchange between the two boards. A driving point has been to test and evaluate various low-level communication data buses to either replace or backup the SpaceWire-Ethernet link. In this process, a new version of the software has been developed with the flexibility to use different communication data buses. One of the key tasks accomplished is the redefinition of the packets' structure and more precisely the encoding of the data. In this view, an approach to reduce the size of the data sent has been chosen and most information is directly encoded in hexadecimal values. In parallel, some constraints are added to enable the use of specific characters for communication purposes. This overall encoding is tailored to an Active Debris Removal mission. The ultimate step has been to create a test procedure for the data exchange and compare the performances of these various low-level data buses. The initial results tend to favor the CAN bus as the main fall-back communication channel for its resilience to error and promising speed. This data bus is also very suited for a cheaper and simpler ADR mission.

   Future work: In this domain, many different aspects need to be worked on to create a robust and resilient data exchange protocol ready for an in-flight demonstration. The first one would be an in-debt analysis of the communication link's performances. It is crucial to assess the limitation and possibilities of the main channel. Moreover, analyzing the delay and the error rate remains a decisive point to achieve a resilient data exchange protocol. In addition, the current work has exclusively looked at computed information being transmitted and did not go on more general data exchange. For the demonstrator mission, it would be significant to allow transmission of raw measurements from the sensor to the PF OBC and thus to the ground control. Furthermore, it would be fundamental for the validity evaluation of the various image processing algorithms. Ultimately, this connection would enable the in-flight tackling of potential issues in the PL OBC. In parallel with the main data link evaluation, the assessment of a potential fall-back connection is essential. Even though the link might not be implemented, it would be judicious to have this protocol well defined.

   The next critical step in the HIL testbench is, of course, the upgrade and addition of elements. For example, having a true payload prototype would allow going further into the validation and testing of the overall avionic. Simultaneously, the addition of various sensors could enable deeper analyses of the data chain. Ultimately, the goal would be to achieve a full flatsat testbench where the majority of the satellite's electronic components

are integrated.

The second part of the thesis approaches the issue of designing a complex and efficient avionic architecture. The goal is to ease the creation and development of such avionics by providing simulation feedback on the general performances. To generate such information, it is key to develop a flexible and reliable simulator adapted to the need of Active Debris Removal missions. Nevertheless, the capabilities of the tool can be applied to other types of spacecraft that require complex architecture. Even though it is primarily made to assist the early design of the ClearSpace-1 satellite, the tool can be employed with space rendezvous and/or crew missions. It is some typical cases where the optimization of the architecture is crucial. With the desire to enhance the autonomous part and the still existing constraint on mass to orbit, this tool can provide an additional design strategy for spacecraft development.

Ultimately, the simulator and optimizer allow to test and explore various options of payload avionic architectures. By implementing complex systems and applying constraints related to physical properties as well as mission requirements, trade-offs and design choices can be made. The multiple results of the optimizer provide insight into a range of avionic parameters and their optimal arrangement. By generating a precise analysis of hardware configuration, the design phase of the ClearSpace-1 satellite PL OBC can be quickened and thus the space debris problem will be tackled faster.

3. In the first iteration, my initial task has been to develop a simulation tool to provide trade-offs. The decisive aspect is to simulate a payload avionic architecture over time and be able to analyze the general performance. To achieve this, the simulator needs to consider the entire payload avionic architecture as well as specific elements such as the sensors or the algorithms. To extend the capabilities of the tool, it has been decided to include pre-processing units as well as the main processor hardware. An essential point to recognize is the tool focuses on the payload computer plus its sensors and not the full avionic architecture of the satellite. The decision has been made since the PL OBC has the most complicated and resource-intensive tasks. The aim is to rapidly establish a baseline for the design and have early inputs on the feasibility. For this reason, the models employed for the various components generally have a high level of abstraction. Furthermore, this tool is not designed to replace the testing phase of actual hardware. A specificity of the models is the implementation of modes for various elements. The idea is that a hardware component such as a sensor can include multiple modes. For example, a camera can be turned off or generating dozens of images per second. This mode concept remains the baseline regarding the state of the system in the whole thesis. The general output of the initial tools is the behavior of multiple elements over time. The simulator works by using a timeline of the modes for all the components in the avionic. It then displays the CPU load inside the main processor during the simulation as well as the memory usage. It also outputs the electric energy consumption of the components and the saturation of the various communication lines. These results allow the creation of trade-offs and enable the testing of multiple configurations.

4. The second iteration of the tool allowed me to go more in-depth in the optimization of the avionic design. To achieve this, the tool is completely rewritten to enable the use of optimization. The key task has been to describe the global payload avionic problem in a mathematical way to allow its implementation in a solver. This version aims to employ the Mixed-Integer NonLinear Program (MINLP) definition. The challenge is to express the relation and behavior of each element with equations. In addition, these equations should be flexible enough to accommodate various numbers of elements. It is essential to allow the user to change the number of sensors or their properties without the need of adjusting the implemented mathematical model. Moving to the optimization world also means the addition of some parameters and main variables. One of the primary aspects is the output of the algorithms in the payload avionic. It has been decided that the fundamental way to evaluate each design would be through the accuracy output of its algorithms. Indeed, the primary goal of the PL OBC is to extract meaningful and accurate information from its various sensors and provide them to the PF OBC. For this reason, an abstract parameter representing the accuracy of these algorithms is created. The optimizer will try maximizing this parameter. Regarding the output, the tool still follows a similar approach as in the previous version. In addition, it provides the modes of the various elements as well as the evolution of the accuracy parameters. A significant point to mention is the modes of all the components are the decisive parameters the optimizer can change. The rest of the variables are here to define the physical constraints of the system. The substantial advantage of this version remains its ability to produce an overall grade to the suggested payload avionic architecture with the evolution of the accuracy parameters.

5. The third iteration of the tool focuses on parametric analyses. My idea was to evaluate the performance of the optimizer by slightly varying one constraint or variable in a standard architecture. To achieve this, the mathematical model of the tool had to be switched to a Mixed-Integer Quadratically Constrained Quadratic Program (MIQCQP) definition. This new model also enabled some improvement regarding the overall optimization process and the extension of a few features such as the accuracy and the mode switching. The evaluation of parameters focuses on their effect on the system's general behavior. By extension, it is primarily on the accuracy over time produced by the algorithms. The general conclusion is that electrical energy remains the leading driver for the architecture's performance, and slight variation could lead to drastic changes. It is a concerning result that needs deeper analysis. The current version of the optimizer enables deeper analyses of various complex payload avionic architectures. Its flexibility allows the study of parameters' effects and correlations as well as simulation constraints. The ease of implementation for complex systems and multiple constraints permits the modeling of a broad range of payload avionics. It enables the extraction of preliminary requirements, trade-offs, and design choices.

6. In my last work regarding the tools development, the idea has been to tackle the verification of the mathematical model. The previous developments were focused on various

aspects of the tool to enhance its ability to provide feedback. One of the critical concerns has been about the verification of the outputted results and their relevance for actual designs. To address the issue, the first task has been to create an Hardware-In-the-Loop setup and run various mock-up algorithms in it. For this initial assessment, the focus has been on the algorithms and their general behavior. The goal is to emulate the load of complex image processing algorithms in a payload computer prototype. By monitoring the CPU load and comparing the results with the optimizer's output, some conclusion has been drawn. The general results indicate the practical model implementation of the algorithm is correct. Even though small issues exist related to the timing and the exact resource consumption, the optimizer follows the same trend as the concrete elements. It is a significant result since this setup validates a part of the optimizer.

7. To conclude the optimization part of the thesis, a realistic scenario has been tested with plausible payload avionic architecture. The goal was to demonstrate the capability of the tool and present the procedure to employ it. In this chapter, typical instruments of an ADR mission have been analyzed. With the result of this investigation, models of actual hardware elements have been implemented in the tool. The PL OBC has also been inspired by an actual electronic board. Regarding the scenario, it has been elaborated to match the constraint of a fly-around mission phase around a target. The output of the simulator shows some interesting behavior such as the limitation in one of the data types due to several factors. The analysis of the output has shown the process to elaborate solutions to improve the payload avionic and better meet the requirements of the mission phase.

   Future work: Despite significant results, the tool could nevertheless require some improvement and additional features. One of the most prominent ones would be regarding its general implementation. Some fundamental aspects such as the number of variables and the design of the mode could be improved. Another critical constraint has been the duration needed for each optimization run. The complexity and thus the solving speed is tightly linked to the number of steps in the model. If an optimization test is required to run over an extensive period of time, the solver will need a consequent amount of time. In addition, some aspects of the constraints could be modified to better handle the initial step of the optimization.

   A crucial part that has already been talked about is the general level of abstraction for the elements. During this thesis, the tool has been mostly used with highly abstract sensors and components that could be far from the real hardware parts. Some additional work could be performed to define better models and therefore creates simulations closer to an actual hardware setup. In the same line, it would be essential to extract constraints from real flight phases of the satellite other than the fly-around. It would enable more realistic scenarios. Nevertheless, this tool is primarily designed for initial assessments of payload avionic architectures, and it makes the abstraction aspect less critical overall.

Following on the same topic, chapter 6 *introduces the first step toward verification of the optimizer. Nevertheless, the sensors have been left out of this initial work and some verification could be required. By merging a model improvement and the implementation of additional hardware elements, some comparison could be performed. It would permit the assessment of the mathematical models of both the sensors and the memories. Furthermore, an interesting study would be to compare the optimizer result with current space missions. It would provide additional feedback and allow to refine the tool.*

As mentioned in the last chapter, the tool developed has some issues with the way to optimize the payload avionic architecture. It would probably be more efficient to task algorithms run-time among multiple orbits instead of doing some compromise in a limited period. Indeed, the compromise option imposes more constraint on the system and does not provide better results. Nevertheless, it is important to remember that this solution might not be possible for every mission phase.

The last general comment is regarding the distribution and usage of the tool. Right now, defining a payload avionic architecture model is done through the writing of an XML file. An ideal solution would be the design of a graphical user interface to ease the creation of these architectures. The interface could be similar to the diagram shown in Figure 5.1 and would allow the user to drag and drop various elements. This feature would make the tool more accessible and appealing to use.

Overall, the design of the avionic for an Active Debris Removal mission has shown to be a complex topic. It regroups the constraints of space missions, rendezvous operation, limited energy and CPU load, and the uncooperativeness of the target. In addition, restrictions due to potential limited ground converge, and the unpredictable state of the debris implies another layer of redundancy. The goal of this thesis has been to address these constraints with two key projects. The first one is the link assessment between the Payload On-Board Computer and the Platform On-Board Computer. The connection is crucial to keep the satellite informed of the general situation and allows it to react accordingly. For the second one, extensive work has been performed to facilitate and assist the development of the Payload On-Board Computer with an optimization tool. All these tasks have been done in the sole optic to accelerate the development of Active Debris Removal missions and therefore preventing a catastrophic Kessler Syndrome.

Appendix Part III

A Hardware-In-the-Loop Testbench

The goal of this appendix is to quickly describe the procedure and software used with the HIL testbench at the EPFL Space Center.

1 GR712RC Switch Matrix for I/O

A key point to mention about the development board is the importance of the Switch Matrix for I/O. A full description can be found in the user manual of the board. This hardware matrix is crucial to use various data buses and enable/disable some functionalities.

2 Compiler

Multiple compilers can be used for the software dedicated to the GR712RC board. In this thesis, the RTEMS LEON/ERC32 Cross-Compiler System is used.

To compile the hello world example, the following command is used :

"sparc-rtems-gcc -g -O2 rtems-hello.c -o rtems-hello"

3 GRMON2

The goal of this tool is to upload software to the processor board (GR712RC). It can be downloaded on the official CAES (Cobham Advanced Electronic Solutions) website but it requires a hardware key to use it. The official documentation can also be found on the same website.

To run the tools, the following command is used :

"grmon -jtag -ftdi -nosram"

Once the command is launched, the user can upload code, verify it and run it. "load testSW"

"verify testSW" "run"

For more advanced software to upload, it is possible to create a ".batch" file and have the instruction for GRMON written in it.

The "info sys" command is useful to see the current module attached to the processors and their status.

To close GRMON, the command "exit" is used.

4 Monitoring

In this development workstation, the minicom program is used.

To run it, the command "minicom -D /dev/ttyS0" is used. The user needs to verify that the UART cable is effectively connected to ttyS0. In some cases, the baud rate needs the be changed such as here: "minicom -D /dev/ttyS0 -b 38400".

B Avionic Simulator

1 Availability

The entire code used for the avionic simulator is available on GitHub (https://github.com/Michael-Juillard/AvionicSimulator).

2 README

2.1 Project Description

The goal of this simulator is to emulate at a high level the principle of a satellite avionic dedicated to Active Debris Removal. It uses profiles created by the users and models of elements to simulate the behavior of the system. More information can be found in the following publication: "Simulation Tool to Study High Performance Avionic for Active Debris Removal Missions" (Link) by Michaël Juillard, Muriel Richard-Noca & Jean-Paul Kneib (2019 IEEE/AIAA 38th Digital Avionics Systems Conference).

The tool is exclusively using Matlab (version 2020a) with no additional libraries. It uses CSV and XML files as the basis for the model and behavior.

2.2 How to Use

This tool requires Matlab to run. To test the basic case, you open the `main.m` file and run it. It will automatically load the needed components.

The "component" folder contains the models of the system. New components can be added easily and should use the structure of the "`testXXX.xml`" file.

The "core" folder has the Matlab code to load the models and behavior of the systems. These files can be modified to include additional functionalities.

The "display" folder contains two standalone scripts to plot the data generated by the tool.

The "profile" folder has descriptions of all the elements' behavior. These files are dictating the variation in the simulation. New profiles can be added easily and should use the structure of the "`testXXX.csv`" file.

The "result" folder will store the results of the simulator.

2.3 Credits

Michael Juillard

C Avionic Optimizer

1 Availability

The entire code used for the last version of the avionic optimizer is available on GitHub (https://github.com/Michael-Juillard/AvionicOptimizer).

2 README

2.1 Project Description

The goal of this optimizer is to find the best avionic architecture for a satellite dedicated to Active Debris Removal. It uses a MIQCQP (Mixed Integer Quadratically-Constrained Quadratic Programming) approach to implement the structure of the avionic. More details on the implementation can be found in part 3 of this readme.

2.2 How to Use

This tool requires Matlab to run. In addition, it needs the Gurobi Matlab library (version 9.1.1) that can be obtained for free with an academic license. The link to the library should be updated in the first line of the `main.m` file.

The "`main.m`" file contains the implementation of the MIQCQP into the library. It is also the one running the whole optimization. The "`loadParamXML.m`" load the various parameter of the architecture from the XML file stored in the "parameter" folder.

The "display" folder contains two standalone scripts to plot the data generated by the tool.

The "parameter" folder contains the description of the avionic architecture that should be optimized. The new architecture can be added easily and should use the structure of the "`param_test.xml`" file.

The "display" folder contains the models of the system. New components can be added easily and should use the structure of the "`testXXX.xml`" file.

The "parametricAnalyses" folder has the Matlab code to load the parametric analyses. It is the same code as the main.m with a few additional links to the parameter used. (In a future version, it will simply call the `main.m` file.)

The "results" folder will store the results of the optimizer.

2.3 Research linked

Complement information on the tool can be found here

- "Simulation Tool: Resources Management in High Performance Avionic for ADR Missions" by Michaël Juillard & Jean-Paul Kneib (2021 IEEE Aerospace Conference)[153]
- "Optimal Control Approach for Dedicated On-Board Computer in Active Debris Removal Mission" by Michaël Juillard & Jean-Paul Kneib (35th Proceedings of the AIAA/USU Conference on Small Satellites)[154]

2.4 Credits

Michael Juillard

Bibliography

[1] C. Bonnal and D. S. McKnigh, "IAA Situation Report on Space Debris," IAA, Tech. Rep., 2017.

[2] ESA, "ESA's Annual Space Environment Report," ESA Space Debris Office, Tech. Rep., May 2021.

[3] N. L. Johnson, E. Stansbery, J. C. Liou, M. Horstman, C. Stokely, and D. Whitlock, "The characteristics and consequences of the break-up of the Fengyun-1C spacecraft," *Acta Astronautica*, vol. 63, no. 1, pp. 128–135, Jul. 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576507003281

[4] L. Anselmo and C. Pardini, "Analysis of the consequences in low Earth orbit of the collision between Cosmos 2251 and Iridium 33," in *Proceedings of the 21st International Symposium on Space Flight Dynamics*. Centre nationale d'etudes spatiales Paris, France, 2009, pp. 1–15.

[5] D. J. Kessler, N. L. Johnson, J. C. Liou, and M. Matney, "The kessler syndrome: implications to future space operations," *Advances in the Astronautical Sciences*, vol. 137, no. 8, p. 2010, 2010.

[6] E. L. Christiansen, *Meteoroid/debris shielding*. National Aeronautics and Space Administration, Lyndon B. Johnson Space, 2003.

[7] F. Schäfer, M. Lambert, E. Christiansen, S. Kibe, H. Stokes, H.-G. Reimerdes, S. Meshcheryakov, F. Angrilli, and H. Zengyao, "The inter-agency space debris coordination committee (IADC) protection manual," in *4th European Conference on Space Debris*, vol. 587, 2005, p. 39.

[8] C. Wiedemann, S. Flegel, M. Möckel, J. Gelhaus, V. Braun, C. Kebschull, J. Kreisel, M. Metz, and P. Vörsmann, "Cost estimation of active debris removal," in *63rd International Astronautical Congress*, vol. IAC-12.A6.5.3. International Astronautical Federation Naples, Italy, 2012.

[9] C. P. Mark and S. Kamath, "Review of active space debris removal methods," *Space Policy*, vol. 47, pp. 194–206, 2019, publisher: Elsevier.

[10] J. L. Forshaw, G. S. Aglietti, N. Navarathinam, H. Kadhem, T. Salmon, A. Pisseloup, E. Joffre, T. Chabot, I. Retat, and R. Axthelm, "RemoveDEBRIS: An in-orbit active debris removal demonstration mission," *Acta Astronautica*, vol. 127, pp. 448–463, 2016.

[11] R. Biesbroek, L. Innocenti, A. Wolahan, and S. M. Serrano, "e. Deorbit-ESA's active debris removal mission," in *Proceedings of the 7th European Conference on Space Debris*, vol. 10. ESA Space Debris Office, 2017.

[12] T. Debus and S. Dougherty, "Overview and performance of the front-end robotics enabling near-term demonstration (FREND) robotic arm," in *AIAA Infotech@ Aerospace Conference and AIAA Unmanned. Unlimited Conference*, 2009, p. 1870.

[13] C. Blackerby, A. Okamoto, K. Fujimoto, N. Okada, J. L. Forshaw, and J. Auburn, "ELSA-d: An In-Orbit End-of-Life Demonstration Mission," in *69th International Astronautical Congress*, 2018.

[14] G. Personne, "ATV GNC synthesis: overall design, operations and main performances," in *Guidance, Navigation and Control Systems*, vol. 606, 2006.

[15] J. Fabrega, M. Frezet, and J.-L. Gonnaud, "ATV GNC during rendezvous," in *Spacecraft Guidance, Navigation and Control Systems*, vol. 381, 1997, p. 85.

[16] S. Persson, S. D'Amico, and J. Harr, "Flight Results from PRISMA Formation Flying and Rendezvous Demonstration Mission," in *61st International Astronautical Congress*, Prag, 2010. [Online]. Available: http://elib.dlr.de/77660/

[17] G. Gaias, J.-S. Ardaens, and C. Schultz, "The AVANTI experiment: flight results," *10th International ESA Conference on Guidance, Navigation & Control Systems*, 2017.

[18] J. Gonnaud, J. Sommer, and M. Tsang, "APRK GNC Design and Performances Evaluation for ATV Rendezvous," in *Proceedings of the 3rd International Conference on Spacecraft Guidance, Navigation and Control Systems*, 1996, pp. 26–29.

[19] D. Pinard, S. Reynaud, P. Delpy, and S. E. Strandmoe, "Accurate and autonomous navigation for the ATV," *Aerospace Science and Technology*, vol. 11, no. 6, pp. 490–498, Sep. 2007. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1270963807000624

[20] L. Blarre, N. Perrimon, C. Moussu, P. Da Cunha, and S. Strandmoe, "Atv videometer qualification," in *55th International Astronautical Congress of the International Astronautical Federation, the International Academy of Astronautics, and the International Institute of Space Law*, 2004, pp. A–3.

[21] S. Dussy, P. Delaux, and P. Simon, "Navigation sensors architecture for the Automated Transfer Vehicle," *IFAC Proceedings Volumes*, vol. 34, no. 15, pp. 365–371, 2001, publisher: Elsevier.

[22] F. M. Kolb, M. Windmüller, M. Rößler, B. Möbius, P. Casiez, B. Cavrois, and O. Mongrard, "The liris-2 3d imaging lidar on atv-5," in *13th Symposium on Advanced Space Technologies in Robotics and Automation*, 2015, p. 1.

[23] M. Ganet, I. Quinquis, J. Bourdon, and P. Delpy, "ATV GNC during rendezvous with ISS," in *DCSSS Conference*. Citeseer, 2002.

[24] D. Berthelier, M. Chaize, P. Delpy, and S. Strandmoe, "Automated Transfer Vehicle (ATV) nominal software GNC numerical validation overview," in *Guidance, Navigation and Control Systems*, vol. 606, 2006.

[25] O. Durou, V. Godet, L. Mangane, D. Pérarnaud, and R. Roques, "Hierarchical fault detection, isolation and recovery applied to COF and ATV avionics," *Acta Astronautica*, vol. 50, no. 9, pp. 547–556, 2002, publisher: Elsevier.

[26] P. Amadieu and J. Y. Heloret, "The automated transfer vehicle," *Air & Space Europe*, vol. 1, no. 1, pp. 76–80, 1999, publisher: Elsevier.

[27] eoPortal, "ATV-5 (Automated Transfer Vehicle-5, Georges Lemaître)." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/i/iss-atv-5

[28] ——, "RemoveDebris Mission." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/r/removedebris

[29] P. Bodin, R. Larsson, F. Nilsson, C. Chasset, R. Noteborn, and M. Nylund, "PRISMA: An In-Orbit Test Bed for Guidance, Navigation, and Control Experiments," *Journal of Spacecraft and Rockets*, vol. 46, no. 3, pp. 615–623, May 2009. [Online]. Available: http://arc.aiaa.org/doi/10.2514/1.40161

[30] P. Bodin, R. Noteborn, R. Larsson, and C. Chasset, "System test results from the GNC experiments on the PRISMA in-orbit test bed," *Acta Astronautica*, vol. 68, no. 7, pp. 862–872, Apr. 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576510003097

[31] S. D'Amico, P. Bodin, M. Delpech, and R. Noteborn, "PRISMA," in *Distributed Space Missions for Earth System Monitoring*, ser. Space Technology Library. Springer, New York, NY, 2013, pp. 599–637. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4614-4541-8_21

[32] G. Gaias, J.-S. Ardaens, and T. Terzibaschian, "Paving the Way for Future On-Orbit-Servicing Missions: the AVANTI Experiment," in *25th International Symposium of Space Flight Dynamics (ISSFD)*, 2015, pp. 1–16.

[33] G. Gaias and J.-S. Ardaens, "In-orbit experience and lessons learned from the AVANTI experiment," *Acta Astronautica*, vol. 153, pp. 383–393, Dec. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576517312602

[34] K. Brieß, W. Bärwald, E. Gill, H. Kayal, O. Montenbruck, S. Montenegro, W. Halle, W. Skrbek, H. Studemund, T. Terzibaschian, and H. Venus, "Technology demonstration by the BIRD-mission," *Acta Astronautica*, vol. 56, no. 1, pp. 57–63, Jan. 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576504002826

[35] eoPortal, "BIROS (Bi-spectral InfraRed Optical System)." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/b/biros

[36] G. S. Aglietti, B. Taylor, S. Fellowes, T. Salmon, I. Retat, A. Hall, T. Chabot, A. Pisseloup, C. Cox, A. Mafficini, and others, "The active space debris removal mission RemoveDebris. Part 2: in orbit operations," *Acta Astronautica*, vol. 168, pp. 310–322, 2020, publisher: Elsevier.

[37] G. Lentaris, I. Stratakos, I. Stamoulias, K. Maragos, D. Soudris, M. Lourakis, X. Zabulis, and D. Gonzalez-Arjona, "Project HIPNOS: Case Study of High Performance Avionics for Active Debris Removal in Space," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2017, pp. 350–355.

[38] GMV, "GMW website." [Online]. Available: http://www.gmv.com/en/sectors/space

[39] ——, "NEOGNC-2 CCN-1 IMAGE PROCESSING FP, GNC for NEO – Phase 2 – CCN-1 IP," Jun. 2015.

[40] ——, "TN-17: FINAL REPORT NEOGNC2-IP," GMW, Technical Report, 2015.

[41] Astroscale, "Astroscale's ELSA-d Successfully Demonstrates Repeated Magnetic Capture," Aug. 2021. [Online]. Available: https://astroscale.com/astroscales-elsa-d-successfully-demonstrates-repeated-magnetic-capture/

[42] SpaceNews, "China's Shijian-21 towed dead satellite to a high graveyard orbit," Jan. 2022. [Online]. Available: https://spacenews.com/chinas-shijian-21-spacecraft-docked-with-and-towed-a-dead-satellite/

[43] M. Schmitz, S. Fasoulas, and J. Utzmann, "Performance model for space-based laser debris sweepers," *Acta Astronautica*, vol. 115, pp. 376–383, 2015.

[44] C. Bombardelli and J. Pelaez, "Ion beam shepherd for contactless space debris removal," *Journal of guidance, control, and dynamics*, vol. 34, no. 3, pp. 916–920, 2011.

[45] M. Andrenucci, P. Pergola, A. Ruggiero, J. Olympio, and L. Summerer, "Active Removal of Space Debris: Expanding foam application for active debris removal," *Final Report, ESA, Contract No. 4000101449/10/NL/CBi*, 2011.

[46] C. Bonnal, J.-M. Ruault, and M.-C. Desjean, "Active debris removal: Recent progress and current trends," *Acta Astronautica*, vol. 85, pp. 51–60, Apr. 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576512004602

[47] J. Eickhoff, *The FLP Microsatellite Platform - Flight Operations Manual*. Springer, 2016. [Online]. Available: //www.springer.com/de/book/9783319235028

[48] J. Eickhoff, Ed., *A Combined Data and Power Management Infrastructure*, 2nd ed., ser. Springer Aerospace Technology. Berlin, Heidelberg: Springer Berlin Heidelberg, 2022. [Online]. Available: https://link.springer.com/book/10.1007/978-3-662-64053-1

[49] Cobham, "GR712 Development Board User Manual," Cobham Gaisler AB, Tech. Rep., Mar. 2018.

[50] O. Notebaert, "Research & Technology activities in on-board data processing domain," ESA/ESTEC in Noordwijk, The Netherlands, Oct. 2015.

[51] SSTL, "OBC750 LEO Flight Computer," Surrey Satellite Technology US LLC (SSTL), datasheet, 2010.

[52] SSL, "SSL 1300 Spacecraft Bus for RSDO Applications," Space Systems/Loral (SSL), datasheet, 2016.

[53] OHB, "Satellite Platforms - Medium and Large platforms," OHB, datasheet, 2018.

[54] LM, "LM100 - A Configure-To-Order Core Spacecraft (Bus) Design for Flexible Payload Accommodation and Mission Operations," Lockheed Martin, datasheet, 2016.

[55] B. Tomei, C. Koduru, S. Sichi, K. Suh, T. Ha, and R. Gupta, "Advanced Space Based Network using Ground Based Beam Former," in *29th AIAA International Communications Satellite Systems Conference (ICSSC-2011)*, 2011, p. 8077.

[56] M. Roux and P. Bertheux, "Alphabus: The European Platform for Large Communications Satellites," in *25th AIAA International Communications Satellite Systems Conference (organized by APSCC)*, 2007, p. 3121.

[57] S. Habinc, R. Martins, J. Costa Pinto, and G. Furano, "Data Handling and Processing Unit for Alphabus/Alphasat TDP-8," *DASIA 2011-Data Systems In Aerospace*, vol. 694, p. 46, 2011.

[58] NG, "GEOSTAR™-3," Northrop Grumman, datasheet, 2020.

[59] F. Arenou, C. Babusiaux, F. Chéreau, and S. Mignot, "The Gaia on-board scientific data handling," in *The Three-Dimensional Universe with Gaia*, vol. 576, 2005, p. 335.

[60] G. D. Racca, R. Laureijs, L. Stagnaro, J.-C. Salvignol, and Al., "The Euclid mission design," in *Space Telescopes and Instrumentation 2016: Optical, Infrared, and Millimeter Wave*, vol. 9904. International Society for Optics and Photonics, Jul. 2016, p. 99040O. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/9904/99040O/The-Euclid-mission-design/10.1117/12.2230762.short

[61] eoPortal, "Gaia Astrometry Mission." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/g/gaia

[62] ——, "Euclid — Mapping the geometry of the dark Universe." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/e/euclid

[63] T. Jaeger, W. Mirczak, and B. Crandall, "Cellularized Satellites - A Small Satellite Instantiation that Provides Mission and Space Access Adaptability," *AIAA/USU Conference on Small Satellites*, Aug. 2016. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2016/TS4AdvTech1/7

[64] A. O. Erlank and C. P. Bridges, "A hybrid real-time agent platform for fault-tolerant, embedded applications," *Autonomous Agents and Multi-Agent Systems*, vol. 32, no. 2, pp. 252–274, Mar. 2018. [Online]. Available: https://link.springer.com/article/10.1007/s10458-017-9378-4

[65] P. I. Deffenbaugh, M. Newton, and K. H. Church, "Digital Manufacturing for Electrically Functional Satlet Structures," *International Symposium on Microelectronics*, vol. 2015, no. 1, pp. 000 210–000 215, Oct. 2015. [Online]. Available: http://www.imapsource.org/doi/abs/10.4071/isom-2015-WA15

[66] P. Melroy, L. Hill, E. E. Fowler, R. Hunter, J. Eagen, B. R. Sullivan, P. Will, and J. Palmer, "DARPA Phoenix Satlets: Progress towards Satellite Cellularization," in *AIAA SPACE 2015 Conference and Exposition*. American Institute of Aeronautics and Astronautics, Aug. 2015, p. 4487. [Online]. Available: http://arc.aiaa.org/doi/10.2514/6.2015-4487

[67] A. O. Erlank and C. P. Bridges, "A multicellular architecture towards low-cost satellite reliability," in *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Jun. 2015, pp. 1–8.

[68] ——, "The satellite stem cell architecture," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec. 2016, pp. 1–8.

[69] ——, "Satellite stem cells: The benefits overheads of reliable, multicellular architectures," in *2017 IEEE Aerospace Conference*, Mar. 2017, pp. 1–12.

[70] ——, "Reliability analysis of multicellular system architectures for low-cost satellites," *Acta Astronautica*, vol. 147, pp. 183–194, Jun. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576517302771

[71] C. Carmichael, E. Fuller, P. Blain, and M. Caffrey, "SEU mitigation techniques for Virtex FPGAs in space applications," in *Proceeding of the Military and Aerospace Programmable Logic Devices International Conference (MAPLD)*, 1999, p. C2.

[72] F. Lima, L. Carro, and R. Reis, "Designing fault tolerant systems into SRAM-based FPGAs," in *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, Jun. 2003, pp. 650–655.

[73] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the Optimal Design of Triple Modular Redundancy Logic for SRAM-based FPGAs," in *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2*, ser. DATE '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 1290–1295. [Online]. Available: http://dx.doi.org/10.1109/DATE.2005.229

[74] L. Sterpone and M. Violante, "A new reliability-oriented place and route algorithm for SRAM-based FPGAs," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 732–744, Jun. 2006.

[75] K. S. Morgan, D. L. McMurtrey, B. H. Pratt, and M. J. Wirthlin, "A Comparison of TMR With Alternative Fault-Tolerant Design Techniques for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2065–2072, Dec. 2007.

[76] B. Pratt, M. Caffrey, J. F. Carroll, P. Graham, K. Morgan, and M. Wirthlin, "Fine-Grain SEU Mitigation for FPGAs Using Partial TMR," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2274–2280, Aug. 2008.

[77] X. Guo-dong, Z. Dan, S. Shi-jie, and L. Sheng-chang, "Run-time control design of micro-satellite OBC based on reconfigurable architecture," in *2009 4th IEEE Conference on Industrial Electronics and Applications*, May 2009, pp. 2591–2595.

[78] B. J. LaMeres and C. Gauer, "Dynamic reconfigurable computing architecture for aerospace applications," in *2009 IEEE Aerospace conference*, Mar. 2009, pp. 1–6.

[79] C. Gauer, B. J. LaMeres, and D. Racek, "Spatial avoidance of hardware faults using FPGA partial reconfiguration of tile-based soft processors," in *2010 IEEE Aerospace Conference*, Mar. 2010, pp. 1–11.

[80] B. LaMeres, S. Harkness, M. Handley, P. Moholt, C. Julien, T. Kaiser, D. Klumpar, K. Mashburn, L. Springer, and G. Crum, "RadSat - Radiation Tolerant SmallSat Computer System," *AIAA/USU Conference on Small Satellites*, Aug. 2015. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2015/all2015/69

[81] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA Design Robustness with Partial TMR," in *2006 IEEE International Reliability Physics Symposium Proceedings*, Mar. 2006, pp. 226–232.

[82] T. M. Lovelly, K. Cheng, W. Garcia, and A. D. George, "Comparative Analysis of Present and Future Space Processors with Device Metrics," in *Proc. of Military and Aerospace Programmable Logic Devices Conference (MAPLD), San Diego, CA*, 2014.

[83] A. Ebrahim, K. Benkrid, X. Iturbe, and C. Hong, "A novel high-performance fault-tolerant ICAP controller," in *2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Jun. 2012, pp. 259–263.

[84] F. Kraja, G. Acher, and A. Bode, "Designing Spacecraft High Performance Computing Architectures," in *Advanced Computing*, ser. Lecture Notes in Computational Science and Engineering. Springer, Berlin, Heidelberg, 2013, pp. 137–156. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-38762-3_7

[85] X. Iturbe, D. Keymeulen, E. Ozer, P. Yiu, D. Berisford, K. Hand, and R. Carlson, "An integrated SoC for science data processing in next-generation space flight instruments avionics," in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, Oct. 2015, pp. 134–141. [Online]. Available: http://ieeexplore.ieee.org/document/7314405/

[86] D. Petrick, N. Gill, M. Hassouneh, R. Stone, L. Winternitz, L. Thomas, M. Davis, P. Sparacino, and T. Flatley, "Adapting the SpaceCube v2.0 data processing system for mission-unique application requirements," in *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, Jun. 2015, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/7231153/

[87] A. Simevski, K. Schleisiek, V. Petrovic, N. Beller, P. Skoncej, G. Schoof, and M. Krstic, "Implementation of a real time unit for satellite applications," in *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, Apr. 2016, pp. 1–6.

[88] D. Rudolph, C. Wilson, J. Stewart, P. Gauvin, A. George, H. Lam, G. Crum, M. Wirthlin, A. Wilson, and A. Stoddard, "CSP: A multifaceted hybrid architecture for space computing," in *Proceedings of the AIAA/USU Conference on Small Satellites*, 2014.

[89] C. Wilson and A. George, "CSP hybrid space computing," *Journal of Aerospace Information Systems*, vol. 15, no. 4, pp. 215–227, 2018, publisher: American Institute of Aeronautics and Astronautics.

[90] A. Williams, J. Puig-Suari, and M. Villa, "Low-cost, low mass avionics system for a dedicated Nano-Satellite launch vehicle," in *2015 IEEE Aerospace Conference*, Mar. 2015, pp. 1–8.

[91] D. Ohlsson, H. Löfgren, E. Vinterhav, and S. Strålsjö, "Enabling advanced missions on small platforms through designing cost effective SpaceWire-based avionics solutions in the CubeSat form factor: SpaceWire missions and applications, short paper," in *2016 International SpaceWire Conference (SpaceWire)*, Oct. 2016, pp. 1–5.

[92] J. J. Lugo and A. Zell, "Framework for autonomous on-board navigation with the AR. Drone," *Journal of Intelligent & Robotic Systems*, vol. 73, no. 1, pp. 401–412, 2014, publisher: Springer.

[93] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, no. 1, pp. 21–39, 2012, publisher: Springer.

[94] DJI, "DJI - Official Website." [Online]. Available: https://www.dji.com/

[95] Parrot, "Parrot - Official Website." [Online]. Available: https://www.parrot.com/en

[96] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 6235–6240.

[97] B. Fuller, J. Kok, N. Kelson, and F. Gonzalez, "Hardware design and implementation of a MAVLink interface for an FPGA-based autonomous UAV flight control system," in *Proceedings of the 16th Australasian Conference on Robotics and Automation 2014*. Australian Robotics and Automation Association Inc., 2014, pp. 1–6.

[98] J. Sandino, F. Vanegas, F. Gonzalez, and F. Maire, "Autonomous uav navigation for active perception of targets in uncertain and cluttered environments," in *2020 IEEE Aerospace Conference*. IEEE, 2020, pp. 1–12.

[99] M. Richard, L. G. Kronig, F. Belloni, V. Gass, O. A. Araromi, H. Shea, C. Paccolat, and J.-P. Thiran, "Uncooperative rendezvous and docking for MicroSats," in *6th International Conference on Recent Advances in Space Technologies, RAST 2013*, 2013.

[100] ClearSpace, "ClearSpace One - A Mission to make space sustainable." [Online]. Available: https://clearspace.today/

[101] M. Juillard, M. Richard-Noca, and J.-P. Kneib, "Dedicated On-Board Computer for Active Debris Removal Mission," in *34th Proceedings of the AIAA/USU Conference on Small Satellites*, 2020, issue: SSC20-P2–17.

[102] J. Eickhoff and et al., "The Flexible LEO Platform for advanced NewSpace Applications," Helsinki, Aug. 2021.

[103] C. Kirlin, "An Introduction to Hardware In the Loop (HIL) Simulation and Its Applications," *International journal of advances in engineering*, vol. 2, no. 1, 2016.

[104] S. Corpino and F. Stesina, "Verification of a CubeSat via hardware-in-the-loop simulation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 50, no. 4, pp. 2807–2818, 2014, publisher: IEEE.

[105] K. Gaber, M. B. El_Mashade, and G. A. A. Aziz, "Hardware-in-the-loop real-time validation of micro-satellite attitude control," *Computers & Electrical Engineering*, vol. 85, p. 106679, 2020, publisher: Elsevier.

[106] T. Chang, "Dynamic characteristics rebuilding of hil simulation system and its validation," in *2010 International Conference on Digital Manufacturing & Automation*, vol. 1. IEEE, 2010, pp. 581–584.

[107] L. SLAFER, "Use of hardware-in-the loop simulation for spacecraft mission preparation, planning and support," in *Aerospace Design Conference*, 1993, p. 1047.

[108] J. C. Gamazo-Real and J. R. Z. Flores, "Integration of COTS Processing Architectures in Small Satellites for Onboard Computing using Fault Injection Testing Methodology," in *3rd International Workshop on Automated and verifiable Software sYstem DEvelopment (ASYDE 2021)*, Jun. 2021.

[109] A. Himmler, J. Allen, and V. Moudgal, "Flexible Avionics Testing-From Virtual ECU Testing to HIL Testing," SAE Technical Paper, Tech. Rep., 2013.

[110] H. A. Rumann, "Advanced Spacecraft Systems Emulation for Space Environment Qualification Testing," in *7th AIAA Atmospheric and Space Environments Conference*, 2015, p. 3321.

[111] D. Jung and P. Tsiotras, "Modeling and hardware-in-the-loop simulation for a small unmanned aerial vehicle," in *AIAA Infotech@ Aerospace 2007 Conference and Exhibit*, 2007, p. 2768.

[112] J. Eickhoff, *Onboard computers, onboard software and satellite operations: an introduction*. Springer Science & Business Media, 2011.

[113] T. Izumisawa and A. Wakatsuki, "A compact and high-speed on-board data bus system using LSIs (satellite communications)," in *IEEE International Conference on Communications,-Spanning the Universe*. IEEE, 1988, pp. 510–515.

[114] B. Palmintier, C. Kitts, P. Stang, and M. Swartwout, "A Distributed Computing Architecture for Small Satellite and Multi-Spacecraft Missions," *Small Satellite Conference*, Aug. 2002.

[115] M. R. Mughal, D. Roascio, and L. M. Reyneri, "Data communication bus interface for modular architecture of small satellites," in *2012 IEEE First AESS European Conference on Satellite Telecommunications*. IEEE, 2012, pp. 1–6.

[116] B. Li and Y. Liu, "Data bus of on-board data system for micro-small satellite," in *Second International Conference on Space Information Technology*, vol. 6795. International Society for Optics and Photonics, 2007, p. 67950K.

[117] L. Zhou and J. An, "Design of a payload data handling system for satellites," in *2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control*. IEEE, 2013, pp. 1217–1220.

[118] wikipedia, "The Open Systems Interconnection model (OSI model)." [Online]. Available: https://en.wikipedia.org/wiki/OSI_model

[119] ECSS, "ECSS-E-ST-50-51C – SpaceWire protocol identification," ECSS, Tech. Rep., May 2010.

[120] ——, "ECSS-E-ST-50-12C Rev.1 – SpaceWire – Links, nodes, routers and networks," ECSS, Tech. Rep., May 2019.

[121] ——, "ECSS-E-ST-50-52C – SpaceWire – Remote memory access protocol," ECSS, Tech. Rep., May 2010.

[122] eoPortal, "Sentinel-1." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/c-missions/copernicus-sentinel-1

[123] ——, "James Webb Space Telescope." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/j/jwst

[124] NASA, "The Neil Gehrels Swift Observatory." [Online]. Available: https://swift.gsfc.nasa.gov/

[125] eoPortal, "BepiColombo." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/b/bepicolombo

[126] ——, "UWE-2 (University of Wurzburg Experimentalsatellit-2)." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/u/uwe-2

[127] ——, "KySat-1 (Kentucky Satellite-1)." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/k/kysat-1

[128] N. Semiconductors, "UM10204: I2C-bus specification and user manual," NXP Semiconductors, Tech. Rep., Jan. 2021.

[129] eoPortal, "Myriade (CNES Microsatellite Program)." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/m/myriade

[130] ——, "SwissCube." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/s/swisscube

[131] ISO, "ISO 11898-1:2015, Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling," Jan. 2015.

[132] eoPortal, "GIOVE-A (Galileo In-Orbit Validation Element-A)." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/g/giove-a

[133] ——, "SMART-1 (Small Mission for Advanced Research in Technology)." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/s/smart-1

[134] T. I. A. (TIA), "TIA-422, Electrical Characteristics of Balanced Voltage Digital Interface Circuits," Jan. 1994. [Online]. Available: https://standards.globalspec.com/std/771962/tia-422

[135] eoPortal, "STP-S26 (Space Test Program-S26)." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/s/stp-s26

[136] ——, "UNIFORM-1 (University International Formation Mission-1)." [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/u/uniform-1

[137] J. S. Dittrich and E. N. Johnson, "Multi-sensor navigation system for an autonomous helicopter," in *Proceedings. The 21st Digital Avionics Systems Conference*, vol. 2, Oct. 2002, pp. 8C1–8C1.

[138] H. Charara and C. Fraboul, "Modelling and simulation of an avionics full duplex switched Ethernet," in *Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop (AICT/SAPIR/ELETE'05)*, Jul. 2005, pp. 207–212.

[139] X. Li and H. Xiong, "Modelling and simulation of integrated modular avionics systems," in *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, Oct. 2009, pp. 7.B.3–1–7.B.3–8.

[140] H. Salzwedel, "Mission level design of avionics," in *The 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576)*, vol. 2, Oct. 2004, pp. 9.D.2–91.

[141] H. Salzwedel, N. Fischer, and G. Schorcht, "Moving Design Automation of Networked Systems to Early Vehicle Level Design Stages," in *SAE World Congress & Exhibition*, Apr. 2009. [Online]. Available: https://www.sae.org/content/2009-01-1375/

[142] N. Fischer, "Automated validation of minimum risk model-based system designs of complex avionics systems," PhD Thesis, Technischen Universitat Ilmenau, 2017.

[143] J. Eickhoff, "TINA Version 5.0 - Mission Planning for Earth Observation," Astrium GmbH, Earth Observation & Science, Tech. Rep., 2001.

[144] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan, "Mixed-integer nonlinear optimization," *Acta Numerica*, vol. 22, pp. 1–131, May 2013. [Online]. Available: https://www.cambridge.org/core/journals/acta-numerica/article/mixedinteger-nonlinear-optimization/2D0CE8CDA53363A31ADE8689565517BD

[145] J. Kronqvist, D. E. Bernal, A. Lundell, and I. E. Grossmann, "A review and comparison of solvers for convex MINLP," *Optimization and Engineering*, vol. 20, no. 2, pp. 397–455, 2019, publisher: Springer.

[146] J. A. Debardelaben, V. K. Madisetti, and A. J. Gadient, "Incorporating cost modeling in embedded-system design," *IEEE Design & Test of Computers*, vol. 14, no. 3, pp. 24–35, 1997, publisher: IEEE.

[147] K. Nagalakshmi and N. Gomathi, "Criticality-cognizant clustering-based task scheduling on multicore processors in the avionics domain," *International Journal of Computational Intelligence Systems*, vol. 11, no. 1, pp. 219–237, 2018, publisher: Atlantis Press.

[148] M. Schlueter, S. O. Erb, M. Gerdts, S. Kemble, and J.-J. Rückmann, "MIDACO on MINLP space applications," *Advances in Space Research*, vol. 51, no. 7, pp. 1116–1131, 2013, publisher: Elsevier.

[149] A. Billionnet, S. Elloumi, and A. Lambert, "Exact quadratic convex reformulations of mixed-integer quadratically constrained problems," *Mathematical Programming*, vol. 158, no. 1, pp. 235–266, 2016, publisher: Springer.

[150] J. F. Franco, M. J. Rider, and R. Romero, "A mixed-integer quadratically-constrained programming model for the distribution system expansion planning," *International Journal of Electrical Power & Energy Systems*, vol. 62, pp. 265–272, 2014, publisher: Elsevier.

[151] C. S. Khor, B. Chachuat, and N. Shah, "Fixed-flowrate total water network synthesis under uncertainty with risk management," *Journal of cleaner production*, vol. 77, pp. 79–93, 2014, publisher: Elsevier.

[152] M. Juillard, M. Richard-Noca, and J.-P. Kneib, "Simulation Tool to Study High Performance Avionic for Active Debris Removal Missions," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE, 2019, pp. 1–10.

[153] M. Juillard and J.-P. Kneib, "Simulation Tool: Resources Management in High Performance Avionic for ADR Missions," in *2021 IEEE Aerospace Conference (50100)*. IEEE, 2021, pp. 1–12.

[154] ——, "Optimal Control Approach for Dedicated On-Board Computer in Active Debris Removal Mission," in *35th Proceedings of the AIAA/USU Conference on Small Satellites*, 2021, issue: SSC21-VII-05.

[155] ——, "Validation of Optimal Control Design Tool for Dedicated Avionic in Active Debris Removal Mission," in *202 IEEE Aerospace Conference*. IEEE, 2022.

[156] ——, "Active Debris Removal Mission: Backup Data Bus for OBC - Payload Computer Interaction," *Acta Astronautica*, 2022, publisher: Elsevier.

[157] Raspberry, "Raspberry Pi 4 Specification." [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

[158] 4Links, "4Links webiste." [Online]. Available: https://www.4links.co.uk/index.php/portfolio

[159] S. Hegde, "Science Data Preprocessor and IP-based Access in the FLP2 Satellite Testbench," Master's thesis, Technical University of Kaiserslautern, 2017.

[160] wikipedia, "IBM System/4 Pi." [Online]. Available: https://en.wikipedia.org/wiki/IBM_System/4_Pi

[161] Waveshare, "RS485 CAN HAT." [Online]. Available: https://www.waveshare.com/wiki/RS485_CAN_HAT

[162] Zihatec, "RS422 / RS485 HAT for Raspberry Pi." [Online]. Available: https://www.hwhardsoft.de/english/projects/rs485-shield/

[163] U. S. D. of Defense, "MIL-STD-1553." [Online]. Available: https://www.milstd1553.com/

[164] wikipedia, "Binary-coded decimal." [Online]. Available: https://en.wikipedia.org/wiki/Binary-coded_decimal

[165] S. Parkes, "SpaceWire User's Guide," STAR-Dundee, Tech. Rep., 2012.

[166] PIGPIO, "PIGPIO library." [Online]. Available: https://abyz.me.uk/rpi/pigpio/

[167] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[168] Gurobi, "Gurobi Optimization website." [Online]. Available: https://www.gurobi.com/

[169] M. Juillard, "CSO-EPFL-1-0-RDV sensor processing and power requirements," EPFL Space Center, Internal Document, Oct. 2019.

[170] PhotonFocus, "Photon Focus website." [Online]. Available: https://www.photonfocus.com/index.php

[171] TSD, "TSD Space website." [Online]. Available: https://www.tsd-space.it/products/imaging-systems/

[172] OPAL, "OPAL website." [Online]. Available: http://www.neptectechnologies.com/products/opal/

[173] MST, "MILITARY SYSTEMS and Technology website." [Online]. Available: https://www.militarysystems-tech.com/suppliers/thermal-vision-specialists/ulis

[174] Jenoptic, "Jenoptic website." [Online]. Available: https://www.jenoptik.com/products/lidar-sensors-technologies

[175] T. Chabot, K. Kanani, A. Pollini, F. Chaumette, E. Marchand, and J. Forshaw, "Vision-based navigation experiment onboard the removedebris mission," in *GNC 2017-10th International ESA Conference on Guidance, Navigation & Control Systems*, 2017, pp. 1–23.

Michaël Juillard

michael.juillard@alumni.epfl.ch
Rte du Flonzaley 7, 1070 Puidoux, Switzerland
Swiss, 03/03/1992
LinkedIn: https://ch.linkedin.com/in/michaël-juillard
Scholar: https://scholar.google.com/citations?user=TxJvXH4AAAAJ

EDUCATION

2018 – 2022	PhD Thesis in Electrical Engineering at EPFL Space Center, EPFL Switzerland, Director-Prof. Jean-Paul Kneib
2015 – 2017	Master of Science MSc in Electrical Engineering, Electronics and Microelectronics, EPFL Switzerland Specialization in Space Technologies Master Thesis (February to August 2017) at MTU Michigan, Project Supervisor-Prof. Lyon Brad King
2014 – 2015	3rd year Bachelor in Electrical Engineering, UCD Ireland (Exchange program with EPFL) Bachelor project at UCD, Project Supervisor-Prof. John T. Sheridan
2012 – 2014	1st & 2nd year Bachelor in Electrical Engineering, EPFL Switzerland
2011 – 2012	6 months English school (New-Zealand) and 4 months Compulsory Swiss Army with the rank of Corporal

PROFESSIONAL EXPERIENCE

PhD Candidate, EPFL (2018-2022 | 4 years)

"*Uncooperative Rendezvous in Space: Design of an Electronic Architecture for High Performance Avionic with Multi Sensor Input and Intensive Data Rate*". Director Prof. Jean-Paul Kneib. Thesis on Advanced Avionic Architecture for Active Debris Removal Missions at EPFL Space Center in partnership with Airbus DS Friedrichshafen and the start-up ClearSpace. The goal of this thesis is to support the development of a dedicated payload avionic for the ClearSpace-1 satellite. The challenge is to optimize the electronics in terms of data processing and power with constraints due to space radiation and failure recovery.
Fiver papers published and one more pending. Best of Track Paper: IEEE/AIAA 38th DASC.

Doctoral Assistant, EPFL (2018-2022 | 4 years)

Help manage lectures. Included giving classes, organizing the examinations, and correcting them.

- "Space Mission Design and Operations" by Claude Nicollier. Master level, Spring. Since 2019 (4 times)
- "EDX : Space Mission Design and Operations" by Claude Nicollier. Online class, Spring. Since 2019 (4 times)
- "Spacecraft Design and System Engineering" by M. Richard & B. Foing. Master level, Fall. Since 2018 (4 times)
- "General Physics : Mechanics" by Cécile Hébert & Daniele Mari. Bachelor level, Fall. Since 2018 (4 times)

In addition, supervision of 4 projects at Master level and system engineer advisor for the EPFL Rocket Team since 2018.

Scientist, EPFL (2018 | 3 months)

Research on FPGA usage for Active Debris Removal Mission CleanSpace One (CSO).

Master Thesis, Michigan Tech University, USA (2017 | 6 months)

"*Evaluation and improvement of 100 Watts Hall-Effect Thruster*". Master Thesis from February to August 2017. The goal was to evaluate and test the design of a low power consumption Hall-Effect thruster. During the evaluation phase, I looked at the thrust, the ISP, the magnetic field, the plasma density and temperature repartition. These parameters are essential for the efficiency evaluation of the thruster.

Engineer, Panoptic Sàrl (2016 | 2 motnhs)

"*Video Stream Control in Gigapixel Multi-Camera Systems*". Developed an interface FPGA-PC for high data rate application with PCIe protocol. I implemented in VHDL and with Vivado(Xilinx) a communication protocol of 9.5 Gigabits per Second in order to transfer a video stream from 16 cameras of 20 megapixels each.

NBC SPECIALIST, SWISS ARMED FORCES (2012-2018 | 4 months cumulated)

Non-commissioned officer (Group Leader). Defense specialist: Nuclear, Biological, and Chemical. Organize training activities at the company level. Duration: initial 21 weeks then 4 weeks every year.

PUBLICATIONS

2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC) – 1st Author

"Simulation Tool to Study High Performance Avionic for Active Debris Removal Missions"

34th Annual Small Satellite Conference (2020) – 1st Author

"Dedicated On-Board Computer for Active Debris Removal Mission"

2021 IEEE Aerospace Conference – 1st Author

"Simulation Tool : Resources Management in High Performance Avionic for ADR Missions"

35th Annual Small Satellite Conference (2021) – 1st Author

"Optimal Control Approach for Dedicated On-Board Computer in Active Debris Removal Mission"

2022 IEEE Aerospace Conference – 1st Author

"Validation of Optimal Control Design Tool for Dedicated Avionic in Active Debris Removal Mission"

Acta Astronautica (2022) – 1st Author - Pending

"Active Debris Removal Mission: Backup Data Bus for OBC - Payload Computer Interaction"

KEY SKILLS

Engineering	System Engineering, Embedded Systems, Hardware-In-the-Loop, Data Buses, Circuit Design
Space	Mission Design, Orbital Mechanics, Electric propulsion
Simulation tools	Matlab (optimal control), Logicim & STK
Programming	C, C++, ASM, Bash (Linux), FPGA-VHDL & python
Office +Web	LaTex, Microsoft Office, JavaScript, PHP, HTML & CSS

EXTRA-CURRICULAR ACTIVITIES

Scouting	Unit Leader 2008-2015. 5 summer camps organized for 30-50 young people, Group leader of "scouts de Nyon" 2015-2017. Team management of 20 leaders and over 120 young people
Licenses	Jeunesse & Sport A+B. Water rescue licenses: Pool Plus
Sports	Rock Climbing & Bouldering, Mountain Trekking, Glider, Badminton, Windsurf, Alpine Ski

LANGUAGES

French	Native language
English	6 months in New Zealand + 1-year exchange in Dublin + 6 moths in the United States, Level C1-C2
German	High school & secondary school, Level B1